2000-09

THORN: a study in designing a usable interface for a Geo-referenced discrete event simulation

Mack, Patrick V.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

THORN: A STUDY IN DESIGNING A USABLE INTERFACE FOR A GEO-REFERENCED DISCRETE EVENT SIMULATION

by

Patrick Mack

September 2000

Thesis Advisor: Arnold H. Buss
Second Reader: Rudy Darken

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 2000	Master's Thesis

4. TITLE AND SUBTITLE
THORN: a study in designing a usable interface for a geo-referenced discrete event simulation

5. FUNDING NUMBERS

6. AUTHOR(S)
Mack, Patrick, V

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)* This thesis evaluates the usability of THORN: a system for displaying a discrete event simulation model in a geographic information system. THORN was developed to enhance the planning phase of Operational Maneuver from the Sea. The goals of this study were to test the system against usability criteria and provide a benchmark for future testing. The purpose of this analysis was to (1) create a system for viewing discrete event simulations fused with geo-referenced spatial information, (2) determine the system's usability, (3) identify problem areas in the graphical user interface, and (4) provide a proof of concept for incorporating usability in the design of military planning tools. The study's scenario is based on the principles outlined in the white paper Operational Maneuver from the Sea. The study tested whether THORN met the usability objectives of (a) 90% successful tasks completion, (b) ease-of-use ratings of "somewhat easy" or better, and (c) satisfaction ratings of "somewhat satisfied" or better. THORN met all of these usability objectives.

14. SUBJECT TERMS
Discrete-Event Simulation, Java, Modeling and Simulation, Operational Maneuver from the Sea.

15. NUMBER OF PAGES
116

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239.18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited.

THORN: A STUDY IN DESIGNING A USABLE INTERFACE FOR A GEO-REFERENCED DISCRETE EVENT SIMULATION

Patrick V. Mack
Lieutenant, United States Navy
B.S., Oregon State University, 1992

Submitted in partial fulfillment of the
requirements for the degrees of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

and

MASTER OF SCIENCE IN COMPUTER SCIENCE
from the
NAVAL POSTGRADUATE SCHOOL
SEPTEMBER 2000

Author: [redacted]

Patrick V. Mack

Approved by: [redacted]

Arnold H. Buss, Thesis Advisor

[redacted]

Rudy Darken, Second Reader

[redacted]

Richard Rosenthal, Chairman
Department of Operations Research

[redacted]

Dan Boger, Chairman
Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis evaluates the usability of THORN: a system for displaying a discrete event simulation model in a geographic information system. THORN was developed to enhance the planning phase of Operational Maneuver from the Sea. The goals of this study were to test the system against usability criteria and provide a benchmark for future testing. The purpose of this analysis was to (1) create a system for viewing discrete event simulations fused with geo-referenced spatial information, (2) determine the system's usability, (3) identify problem areas in the graphical user interface, and (4) provide a proof of concept for incorporating usability in the design of military planning tools. The study's scenario is based on the principles outlined in the white paper Operational Maneuver from the Sea. The study tested whether THORN met the usability objectives of (a) 90% successful tasks completion, (b) ease-of-use ratings of "somewhat easy" or better, and (c) satisfaction ratings of "somewhat satisfied" or better. THORN met all of these usability objectives.

THIS PAGE INTENTIONALLY LEFT BLANK

DISCLAIMER

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made, within the time available to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF SYMBOLS, ACRONYMS AND/OR ABBREVIATIONS

AOA	Analysis of Alternatives
ASM	Anti-Ship Missile
C^2	Command and Control
C^4I	Command, Control, Communications, Computers, and Intelligence
CEP	Circular Error Probable
DoD	Department of Defense
DP	Departure Point
ELAN	Enhanced Lanchester
FFTS	Forward...From The Sea
GPS	Global Positioning System
GIS	Geographical Information System
GUI	Graphical User Interface
LOD	Line of Debarka
MOE	Measure of Effectiveness
MOP	Measure of Performance
MRSI	Multiple Rounds Simultaneous Impact
MVC	Model-View-Controller
NEF	Naval Expeditionary Force
NGFS	Naval Gun Fire Support
NPS	Naval Postgraduate School
NSFS	Naval Surface Fire Support
OMFTS	Operational Maneuver From The Sea
OOP	Object-Oriented Programming
OPNAV	Office of the Chief of Naval Operations
TACAIR	Tactical Aircraft
TAFSM	Target Acquisition Fire Support Model
TBM	Theater Ballistic Missile
TLAM	Tomahawk Land Attack Missile
TLE	Target Location Error
TOF	Time of Flight
TPM	Technical Performance Measure
TTP	Tactics, Techniques, and Procedures

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Joint Vision 2010 established a strategic direction for leveraging technological opportunities to achieve new levels of effectiveness in joint war fighting capabilities for the United States military. Of JV2010's five areas of focus, Information Superiority is the most affected by emerging computer, communication, micro-miniaturization, and Internet technologies. Information Superiority will play a dominant role in the future of warfare.

The Department of Defense (DoD) has responded to Joint Vision 2010's technological challenges by developing advanced war fighting concepts like Operational Maneuver from the Sea (OMFTS). OMFTS is amphibious assault strategy that uses speed, deception, and communication to overwhelm the opposition. It is highly dependant on the ability to rapidly plan the assault, monitor both opposition and friendly forces, and exploit opposition weaknesses as they appear. This major shift in strategy will need to be extensively studied to ensure its feasibility. The DoD is developing simulation systems to assist in the study and implementation of OMFTS.

The current DoD simulation systems have been characterized as stove-piped monolithic systems that are costly, inflexible, and difficult to configure. These characteristics make these systems unsuitable for use in planning operations like OMFTS. Modern software development has shifted from the design of large, complex systems to small flexible systems that are extremely powerful. The small powerful systems can be used as components for larger systems. This modular design philosophy gives users the ability to just use the applicable portions of the system. It also gives software maintainers the ability to rapidly implement technological advances. The maintainer can simply remove the old software component and insert a new one. The same update on a stove-piped system might require a complete software redesign.

Modern software developers have also recognized the importance of usability. Usability describes the efficiency of the interaction between the human and computer. Highly usable systems are those that require very little user training and enable to user to operate at or near peak efficiency. Operations such as OMFTS require simulation systems

that are highly usable. OMFTS planners must be able to plan and execute the operation extremely quickly. They must discover the opposition's weakness, and maneuver friendly forces in a manner that exploits the weakness.

This thesis develops and evaluates the usability of a Geographical Information System (GIS) designed to support OMFTS mission planning, THORN. THORN is a simulation system that allows discrete event simulations to be viewed as animated layers of a map. The map is unique in that it fuses data collected from various repositories with geographical information. This data fusion gives the analyst the ability to view all pertinent data of a region in a single display. This ability to view all OMFTS relevant data of a region in a single display could greatly benefit planners by decreasing decision cycle time.

THORN is based on an Open Source GIS application called OpenMap™. The use of Open Source software as a foundation for specific applications gives the application developer the ability to incorporate technological advances without incurring commercial upgrade costs. It also gives the developer the ability to tailor the graphic user interface (GUI) to the target users. Schneiderman's (1997) eight golden rules of interface design to maximize usability:

1. strive for consistency
2. enable frequent users to use shortcuts
3. offer informative feedback
4. design dialogs to yield closure
5. offer error prevention and simple error handling
6. permit easy reversal of actions
7. support internal locus of control
8. reduce short-term memory load

The process of measuring the usability of an application is called Usability Engineering. Usability Engineering involves testing the efficiency of the Human Computer Interaction (HCI). A usability test was conducted on THORN to measure the efficiency of the interface, to identify problem areas, and to provide a baseline for future testing. THORN'S usability testing criteria were set as follows:

- 90% Successful completion of tasks.

- 90% Error free rate.
- 90% score of 3 or better on a 7 point scale (e.g., 1=easy, 3=somewhat easy, 5=somewhat difficult, and 7=difficult) in ease-of-use.

The task completion rate, and error free rate were set at 90% due to the nature of the application environment. OMFTS planning requires a very efficient interface. Task completion and error free rates of 80% are not uncommon in commercial application testing. Additionally, it was hypothesized that demographic information could be used to predict a user's ability to determine the function of a button by the icon.

THORN met all usability criteria. However, there is room for improvement. Specifically, the annotation interface should be more consistent and interactive. This portion of the interface had the lowest user satisfaction, but met the objectives for task completion. The results of the analysis of demographic data as predictors for button function were not significant. This may be explained by several factors: the participants used in this study had very similar demographics, this increase in the overall population's computer literacy eliminates the effect of demographic predictors, and the demographic data collected in this experiment did not contain effective predictors.

THORN successfully combines many of the proven tools from GIS software into a streamlined design while incorporating the strong design points of Human Factors guidelines. With continued GUI improvement and testing, THORN can grow to become a powerful and portable map-based mission-planning tool for OMFTS.

ACKNOWLEDGMENT

The author would like to express sincere appreciation to his thesis committee members, Dr. Arnold Buss and Dr.Rudy Darken. Acknowledgement and appreciation is also due to Dr. Gordon Bradley for his acceptance and assistance with THORN.

Very sincere thanks and gratitude is also expressed to Mrs. Tasha Ginet for her love, support, patience, and encouragement through this process. I would also like to thank Riley and Drew Mack for their understanding and continued gift of unconditional love.

I. INTRODUCTION

Joint Vision 2010 (JV2010) establishes a conceptual template for leveraging technological opportunities to achieve new levels of effectiveness in joint war fighting for the United States military (CJCS, 1996). Of JV2010's five areas of focus, Information Superiority is most affected by emerging computer, and software technologies. It is the intent of the Department of Defense to use Information to "mitigate the impact of the friction and fog of war." Information Superiority will simultaneously be used, at the same time, to deny the enemy the right to the same (CJCS, 1996). JV2010 provides overall guidance on the development of new war fighting strategies, and the software used to plan and implement these strategies.

This thesis provides a prototype software-planning tool, THORN for evaluating amphibious operations. THORN was developed in the context of a new military strategy that is heavily reliant on Information Superiority; Operational Maneuver from the Sea.

A. OPERATIONAL MANUVER FROM THE SEA

In the white papers, "...from the Sea" and "Forward ... from the Sea," the United States Naval Forces established a visionary approach to naval operations. The approach shifted National Security interests from the historic oceanic "blue water" activities onto the littoral areas. The vision also establishes a framework for the concept of a naval expeditionary force, and in doing so provides the foundation for Operational Maneuver from the Sea (OMFTS), Figure 1 summarizes the six fundamental principles of OMFTS.

OMFTS springs from a desire to capitalize on the opportunities that be may found in the "chaos in the littoral" – a world characterized by the clash of the myriad forces of national aspiration, religious intolerance, and ethnic hatred. The opportunity is manifested by realizing the significant technological advantage the United States has established, specifically, through enhancements in information management, battlefield mobility, and improved conventional weapon lethality.

Principles of

Operational Maneuver from the Sea

- Focuses on an operational objective.
- Uses the sea as maneuver space.
- Generates overwhelming tempo and momentum.
- Pits strength against weakness.
- Emphasizes intelligence, deceptions, and flexibility.
- All organic, joint, and combined assets.

Figure 1 OMFTS principles

OMFTS assumes that the majority of future threats to United States national security will be associated with the littorals, areas characterized by large coastal cities, concentrated populations, and centers of trade. While these areas make up an insignificant percentage of the world's total landmass, they account for over 80% of the world's capital cities and the majority of the marketplaces for international trade. These characteristics make the littoral the most likely location of important future conflicts.

If America desires to continue to influence global events, it must have a credible, forward deployable power projection capability. This capability should include a force that is independent of forward staging areas, friendly borders, or other politically based constraints. The uncertainty of future conflict dictates that the US maintain a wholly self-sufficient force capable of projecting power ashore against opposition forces and to support our national strategic policy.

OMFTS' defining characteristic is the maneuvering of naval forces at the operational level to exploit a significant enemy weakness in order to deal a decisive blow. Movement that may lead to unproductive or counterproductive results does not qualify as an Operational Maneuver. An Operational Maneuver should be directed against an enemy center of gravity: something that is essential to the enemy's ability to effectively continue the fight. This center of gravity may be physical (units, cities, command structure) or a sustaining entity (logistics system). However, the center of gravity could also be an intangible element of the political and moral forces that bind our enemy in the fight against our forces.

Maneuvering against the enemy is not new or unique; what distinguishes OMFTS is the use of the sea as a movement medium. This characteristic enables amphibious forces to use the sea as a means of gaining advantage and as an avenue for friendly movement that is simultaneously a barrier to the enemy and a means of avoiding tactically unwise engagements. Technologies that make this possible include, but are not limited to, sea-based logistics, sea-based fire support and the use of the sea as a medium for tactical and operational movement.

Command and control systems oriented toward rapid decision-making at all levels of command, give Naval forces the speed and flexibility needed to reduce the decision cycle time to orders of magnitude lower than that of the opposing force. This shortened cycle will give friendly forces the ability to exploit enemy vulnerabilities before they can be corrected. In essence, friendly naval forces will be able to act so quickly that the opposing force will not be able to stave off the attack and will be overwhelmingly defeated.

This major change in strategy must be exercised and evaluated to ensure success when used on the battlefield. The use of computer simulation systems to assist in the evaluation of military strategies is an effective means of identifying potential problems. However, as stated in DoD 5000.59-P, "DoD Modeling and Simulation (M&S) Master Plan", current simulation systems:

- Are narrowly focused, stove-piped developments for each user community.
- Take too long to build.
- Are not interoperable with other M&S assets that could be useful.
- Are not easily maintainable or extensible.

Operations like OMFTS have to be planned, exercised, and executed in a very short period of time. The success of these strategies is directly proportional to the ability to execute operations before the opposition can respond. THORN was developed to address the need for a simulation system that is portable, easily configured, and can be run and reconfigured in a relatively short time.

Current simulation systems do not provide a flexible, portable tool for the analysis of operations like OMFTS. As stated earlier these stove-piped systems do not fully meet the needs of the military, and are not interoperable with other assets that could be useful.

Simulation systems should depict the battle-space as a Geographical Information System (GIS). A GIS is a computer system capable of assembling, storing, manipulating, and displaying geographically referenced information, that is, data identified according to their locations. They should provide a singular, fused view of the environment that gives the analyst access to all of the pertinent information as layers that are created independently of each other, and can be fused by geography. The significant advantage of the GIS type application lies in the ability to gather current data from worldwide repositories, some of which may have no visual component, and display this data in an environment that promotes rapid power analysis, model verification, and validation.

The current trend in DoD simulation development is to provide a singular software solution for all simulation requirements. This approach to software design has proven to be expensive, complex, and extremely difficult to configure. Strategies like OMFTS require simulations systems that can be run on a vast array of computational devices. OMFTS planning and preparation must be conducted at all levels of the force, from the general officer to the sergeant embarked in the amphibious assault ship. This need for readily available planning tools translates to an ability to use the software on computers that range from the hand-held to the supercomputer. Current DoD simulation systems have been developed for the desktop personal computer, or the workstation. These systems, by the inherent complexity of design, can be difficult to configure and update. As shown in the OMFTS scenario in chapter II, this strategy depends on the ability to rapidly deploy forces in a manner that will overwhelm the opposition forces. The decision to deploy the force is made after the engagement has started. This compressed timeline demands a planning tool that can be configured and reconfigured extremely rapidly.

A software tool must be usable to be effective. As the software design discipline matures, the importance of incorporating software usability into the development process has increased. The current DoD simulation system's approach to software design seems to address usability as an afterthought. By specifying a system that is capable of

performing every conceivable task, you also require the user, who may only need to perform relatively simple tasks, to learn how to manipulate this complex system to get a relatively simple solution. This approach runs counter to that of modern software design, which develops small-specialized software products capable of performing relatively small task sets extremely well and integrates them using a component architecture.

Joint Vision 2010 details the rapidly changing aspect of modern warfare. It establishes the need for strategies that can be updated rapidly to leverage advances in technology. Planning tools must have, as part of their design, the ability to be rapidly updated. Large, complex simulation tools that have not been designed as modules, and can be easily replaced with updated technology, cannot meet this requirement for rapid updates.

B. THORN

THORN, see Figure 2, was developed to fuse spatially related data from various repositories with information obtained from a running discrete event simulation. It is a system for rapidly planning and evaluating military operations. THORN addresses the need for a usable simulation application that can be rapidly configured with the best information available, provide real-time information coupled with data obtained from an empirical simulation, and can be run on any available computer. It is these features that make THORN an ideal planning tool for operations like OMFTS. THORN can take non-traditional geographic data, like the output of a discrete event simulation, and place it in a context that facilitates analysis. Its ability to fuse data as layers can allow OMFTS

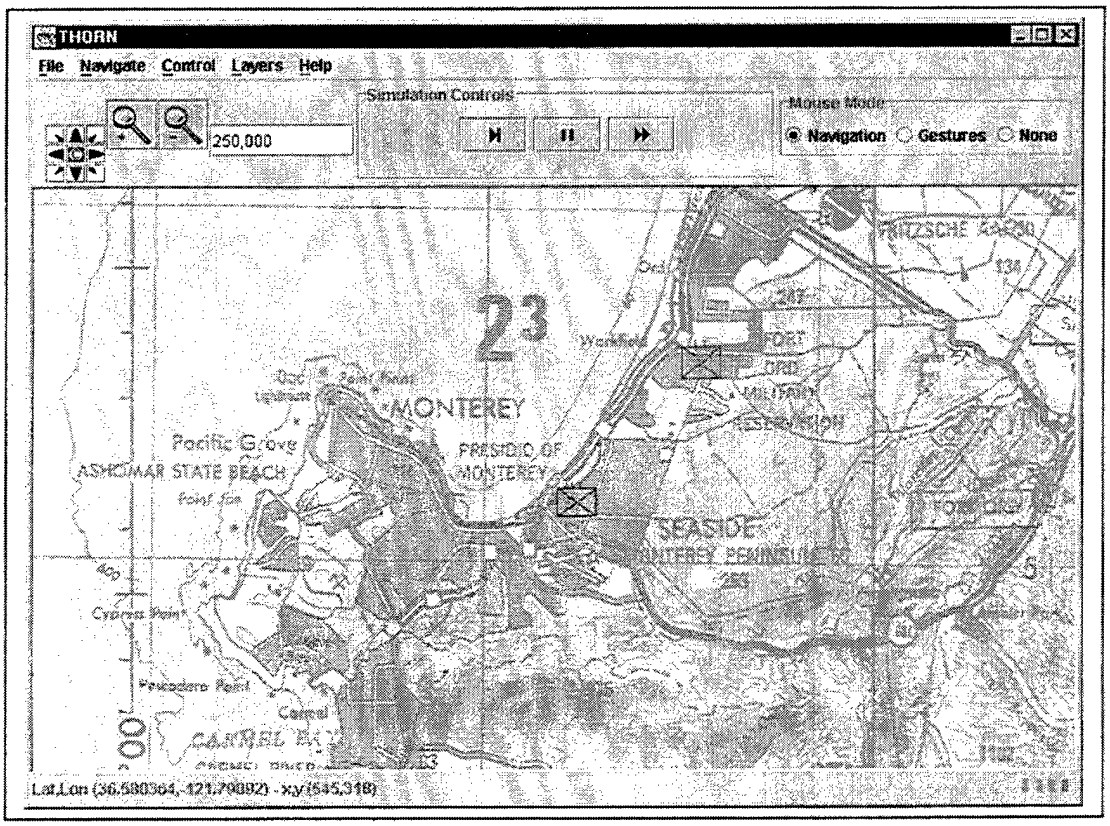


Figure 2 THORN Display

planners to isolate relevant information to assist in the exploitation of the opposition's current posture.

THORN is still in the development phase and will be continually improved based on empirical studies and subject matter expertise. THORN was developed by leveraging open source software, namely OpenMap™. This initial design, version 1.0, is the focus of this study. A usability test was conducted on THORN to evaluate human performance and user preferences. This test also identifies usability issues that focus on future design and redesign efforts.

C. PROBLEM STATEMENT

As previously discussed current military planning systems tend not to be flexible, extensible, or easily configured. More importantly, few military planning systems incorporate usability design practices during initial development. This thesis addresses those problems. Specifically the thesis:

- develops a Graphical Information System that incorporates a discrete event simulation,
- performs baseline usability analysis on the system,
- provides a proof of concept for the development of a functional, highly usable military software-planning tool.

Additionally, the thesis attempts to predict the effectiveness of an interface design by analyzing demographic data.

D. OUTLINE OF THE THESIS

This thesis develops a graphical movement simulation that allows the analyst to visualize the effects of the movement of forces and other data on a geographical display. To avoid the reliance on close, proprietary systems, the software developed in this thesis is built using open source software.

The thesis is organized as follows. Chapter II provides an overview of OMFTS, GIS, usability, and usability metrics. This information is provided as a means of familiarization to the challenges associated with producing a usable tool for working with

and analyzing geo-referenced data. Chapter III details the software architecture of the OpenMap™ system and provides an overview of the additions required to incorporate a discrete event layer, and simulation interface. Chapter III also details all user interface components. Chapter IV details the experimental design of the usability testing experiment, and provides the results of the experiment, and Chapter V summarizes the research, and makes recommendations for future works.

II. BACKGROUND

A. OVERVIEW

Modern software design is transitioning from large, resource-intensive applications, to programs that run on a hand-held computer or desktop machine. As computational power increases, software size and resource demand has decreased. The development of large monolithic solutions has given way to a loosely-coupled suite of smaller modular applications. The Internet revolution has fueled this transition. It is no longer acceptable to produce applications that isolate the user. Information must be shared, and successful applications must foster this sharing effort. Constraints on time to market and the notion of "internet time" – (the speed at which changes propagate on the internet), demands that information be shared in order to succeed.

Internet time has also forced application designers to produce products that have very small learning curves. Users must be able to achieve sustained peak levels of efficiency quickly, in order for the application to succeed in a competitive marketplace. Small, flexible, efficient, and highly usable applications are the modern software ideal. Current DoD simulation systems are the antithesis of the modern software ideal.

DoD has correctly identified the need for simulation systems that can be used jointly. The lack of fiscal resources, the shrinking size of the US military, and the increase in US military obligations demand systems that are capable of sharing information between all forces. The defense department should not select a complex, large, and singular software solution to address this need. They should not choose a system that attempts to address all military planning needs from strategic planning to low intensity conflict. This would be akin to using a 100-pound pipe wrench for all plumbing needs. If the user can configure the application to run in his environment, he must also learn a host of commands that may not be germane to the task.

An additional difficulty in the singular approach is to design an interface for the system that is as equally usable to the soldier in the field as the analyst on the flag officer's staff. Given a "one system approach", it is not possible to produce a usable interface for all needs. Software needs to be made more usable, and usability engineering

requires testing. The participants in the evaluation must be the targeted end-users of the application. The target end-users of the simulation system are every leader in all four US military forces. This interface must be equally usable to US Navy Surface Warfare officers and US Army artillery officers.

Java™ is the language of the Internet, and can be run on the majority of computers available today. These two technologies, Java and the Internet, enable simulation system developers to create small powerful components and test them on the specific group of end-users. These smaller components can then act as building blocks for larger more complex systems, creating a modular design. This modular design allows developers to add or remove components as necessary to update technology without having to redesign the entire system.

The Internet Revolution has changed the way the majority of industries conduct business. The major industries have adapted their practices to embrace this new approach to software design. That is the majority of industries save one, the DoD. The DoD should take advantage of the changes that are taking place globally. The "one-size-fits-all" solution is not feasible. The following section provides an overview of OMFTS by way of example. The planning tool used in the creation of this example was THORN.

B. OMFTS EXAMPLE

In this illustrative example naval expeditionary forces (NEF) respond to the littoral conflict on the western coast of North America. From the line of debarkation (LOD) the force moves to the departure point (DP). In Phase I, shown in, objectives at littoral penetration areas (LPA), San Francisco, Monterey, and San Diego can be struck. The choice of the LPA will be based on vulnerability. The opposing force must protect all LPAs and has exposed himself to deception: a feint attack on a LPA would divert attention from the true objective.

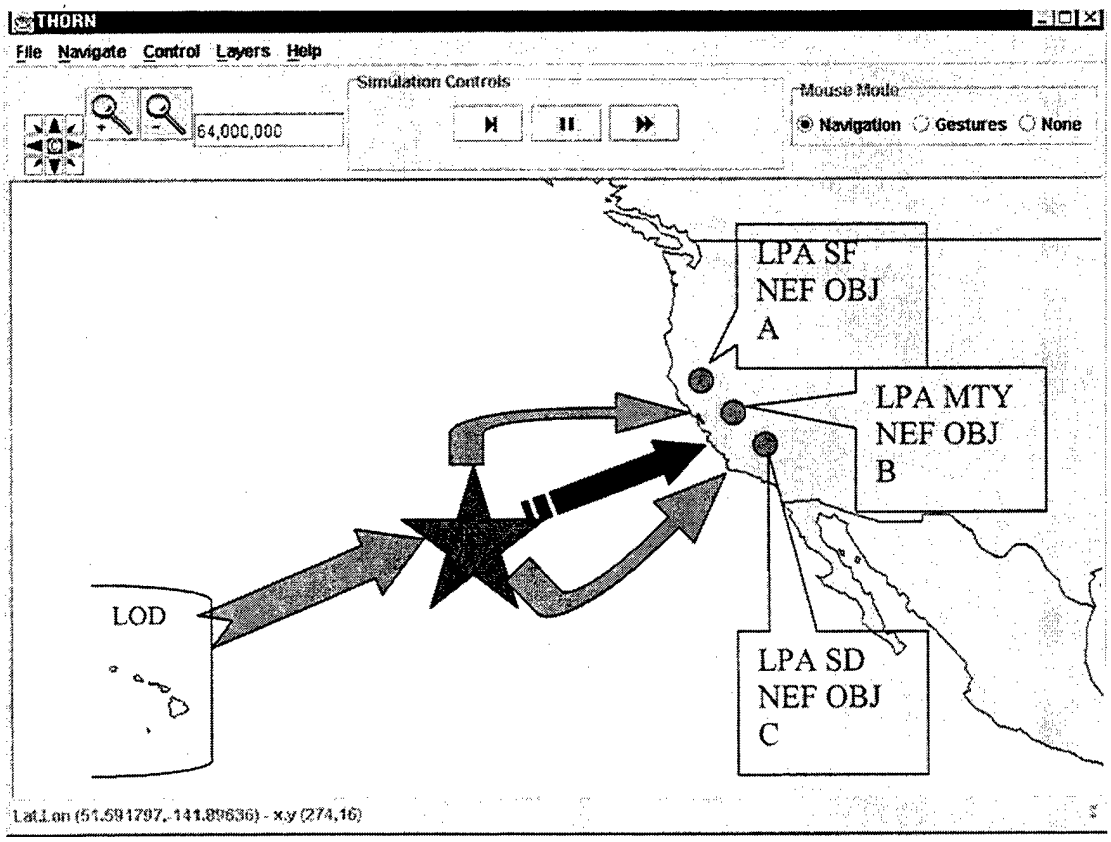


Figure 3 Phase I OMFTS

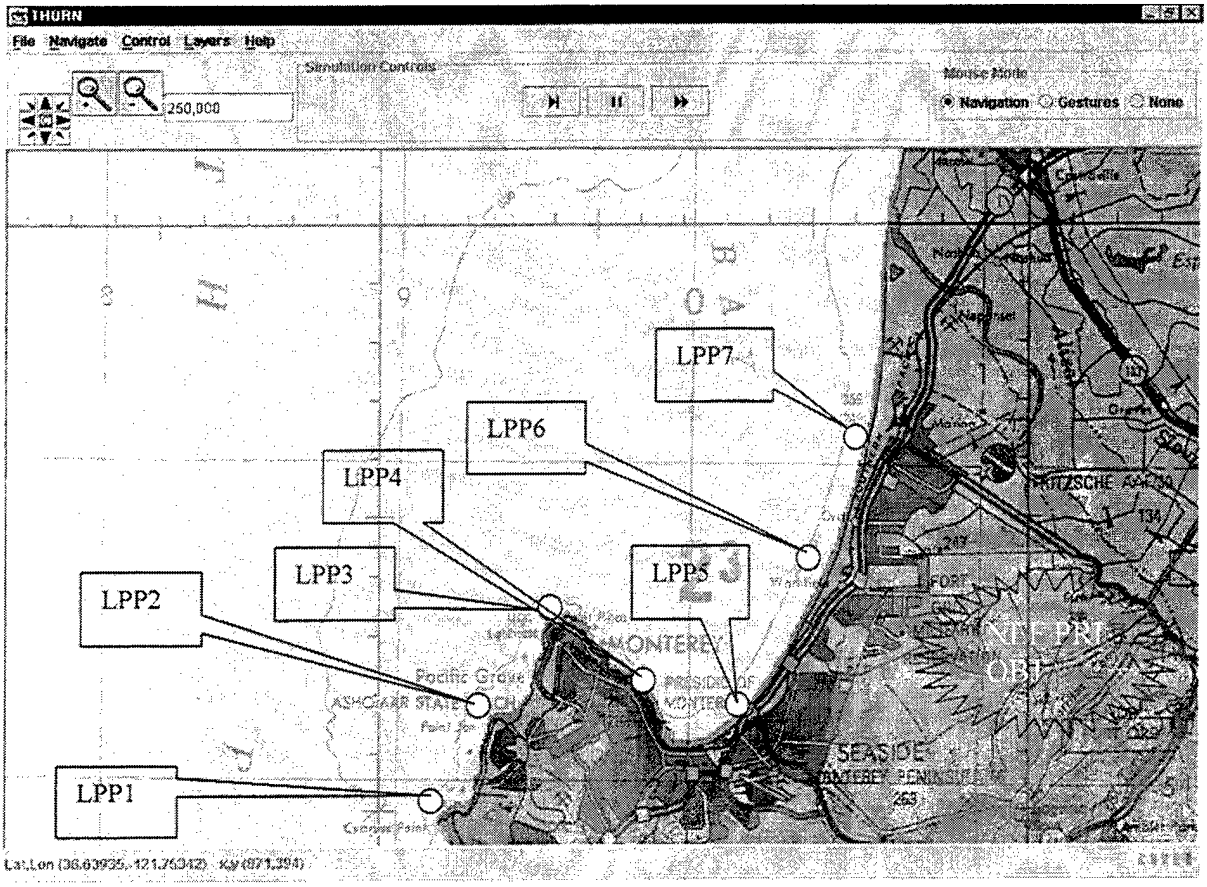


Figure 4 Phase II OMFTS

Phase II, shown in Figure 4, begins with an attack on Monterey, advance operations and real-time reconnaissance identify exploitable littoral penetration points (LPP) through which forces maneuver to overwhelm enemy defenses. The attack transitions from ship to objective without a lengthy buildup of beachheads.

As detailed in JV2010, the war fighter of the future will rely on the use of computer systems to assist in the fight. One such system that could be used in the planning of OMFTS operations is a Geographic Information System, which is discussed in the following section.

C. GIS

A Geographic Information System (GIS) is a computer system capable of assembling, storing, manipulating, and displaying geographically referenced information,

that is, data identified according to their locations. Practitioners sometimes include data such as operating personnel in a GIS. A GIS is both a database with specific capabilities for spatially referenced data, as well as a set of operations for manipulating and analyzing the data. (Star and Estes, 1990).

A GIS stores information about the world as a collection of related layers that can be linked together by geography. This simple, but extremely powerful and versatile concept has proven invaluable for analyzing many real-world problems, such as tracking delivery vehicles, recording details of planning applications, and modeling global atmospheric circulation.

Geographic information may explicitly reference a geographic location, such as a latitude and longitude or national grid coordinate, or it may implicitly refer to such locations as an address, postal code, census tract name, forest stand identifier, or road name. An automated process called geocoding is used to create explicit geographic references (multiple locations) from implicit references (descriptions such as addresses). These geographic references allow the location features, (such as a business or forest stand), and events, (such as an earthquake, on the earth's surface) to be analyzed.

Geographic information systems work with two fundamentally different types of geographic display models--the "vector" model and the "raster" model. In the vector model, information about points, lines, and polygons is encoded and stored as a collection of x,y coordinates. The location of a point feature, such as a borehole, can be described by a single x,y coordinate. Linear features, such as roads and rivers, can be stored as a collection of point coordinates. Polygonal features, such as sales territories and river catchments, can be stored as a closed loop of coordinates. The vector model is extremely useful for describing discrete features, but less so for describing continuously varying features such as soil type or accessibility costs for hospitals. A raster image comprises a collection of grid cells rather like a scanned map or picture. The raster model has evolved to model such continuous features. Modern GISs are able to handle both models.

OMFTS planners could use the features of the modern GIS to look for flaws in the existing infrastructure and exploit them. For example, if the traffic flow along I-95 routinely bottlenecks between LPP6 and LPP4 (from figure 4), then it would be worthwhile to conduct a feint to LPP6. This feint would cause the opposition to mass

forces to defend LPP6. Once the opposition discovered the true LPP, the bottleneck would prevent the rapid redeployment of forces. While simplistic in nature, this type of infrastructure information provided by a GIS coupled with force location data can provide significant opportunities to exploit inherent opposition weaknesses. The GIS application must present this information to the user in an interface that is unambiguous and intuitive. This interface must posses a high level of usability, which is now described.

D. USABILITY

Usability means that the people who use the product are able to do so quickly and easily to accomplish their own tasks (Dumas and Redish, 1994). Usability Engineering is a systematic approach to usability based on four essential points:

1. Focus on users.
2. People utilize products to be productive.
3. People have limited time to accomplish tasks.
4. Users decide when a product is easy to use.

Usability is concerned with the overall utility of the application. Usability should not only be considered an issue for the primary system functionality, but should also be applied to training materials, help packages, and other associated features of the system. In order to improve the ease-of-use of a product, usability should be considered throughout the development of a system, from initial design through final deployment of the system. Dumas and Redish (1994) provide seven principles for ensuring usability:

1. Engineering it into a product through iterative design and development process.
2. Involving users throughout the process.
3. Allowing usability and users' needs to drive design decisions.
4. Working in teams that include skilled usability specialists, interface designers, and technical communicators.
5. Setting quantitative usability goals early in the process.
6. Testing products for usability, but also integrating usability testing with other methods for ensuring usability.
7. Being committed to making technology work for people.

The integration of usability into a product is commonly called "usability engineering", (Good, 1988; Whiteside, Bennett, and Holtzblatt, 1987). Usability engineering, similar to software engineering, includes identifying users, analyzing tasks, setting specifications, developing and testing prototypes, and the iterative cycles of development and testing (Dumas and Redish, 1994). Gould and Lewis (1985) highlight four principles to facilitate designing usability into products.

1. Focus early and continuously on users.
2. Integrate consideration of all aspects of usability.
3. Test versions with users early and continuously.
4. Iterate the design.

Identifying usability requirements prior to design can save time and money for the designer as well as increase the likelihood of user satisfaction with the product. Systems are developed to help individuals accomplish a task. In order to provide a usable system, what the individual needs and how they are to accomplish this must be ascertained. The primary requirement is to understand the prospective users and the audience for a system. Sumas and Redish (1994) have identified techniques that can be used in a usability engineering process. These techniques highlight the importance of describing what a person does in their job in terms of tasks. When the tasks are analyzed, how the person does the job, can do the job, or should do the job are described (Drury, Paramore, Van Cott, Grey and Corelett, 1987). Task analysis will determine whether the correct measurements are performed during the usability analysis.

The aforementioned usability principles are embodied in Shneiderman's (1997) eight "golden rules" of interface design to maximize usability. These are:

1. strive for consistency
2. enable frequent users to use shortcuts
3. offer informative feedback
4. design dialogs to yield closure
5. offer error prevention and simple error handling
6. permit easy reversal of actions
7. support internal locus of control
8. reduce short-term memory load

These eight rules provide a foundation for ensuring usability. Complex systems being developed in support of technical strategies, such as Information Superiority, must incorporate these principles from the onset of development. These usability design principles are critical elements of software applications that support activities like OMFTS. THORN is being developed to provide a foundation application utilizing these principles. The long-term objective is to provide a methodology and baseline for the development of similar systems to support OMFTS, and to ensure that the product remains efficient, effective, and usable.

The OMFTS strategy provides a unique usability challenge. OMFTS requires that users be able to interact with information without it. For example, planners must be able to annotate the display, or highlight areas on the map. Planners must also be able to obtain information from any simulations that may be running in the display. These tasks must be accomplished in a manner that is efficient and prevents the user from inadvertently permanently changing the underlying maps. The application must be able to discriminate the user's desires based on mouse actions and map content alone. Clicking on a simulated unit should provide an output of the unit's disposition, without changing the simulation. While clicking and dragging on the display should draw an annotation object, it should not permanently alter any of the topological information displayed. Mission planners must be able to quickly asses force disposition, both friendly and opposition, and indicate or highlight tactically significant topological information. The application must provide a highly efficient, usable means of accomplishing these tasks. The level of application efficiency can be determined using usability metrics, which are discussed next.

E. USABILITY METRICS

Usability metrics are measurable performance observations that can be used as indicators of an applications' usability. Care must be exercised when selecting metrics to ensure that the chosen metric is an appropriate indicator of usability. The selection of these metrics obviously depends on the goals of the research. In the case of exploratory studies designed to find the source of difficulties and errors, it is best to observe performance in a relevant task in a rather free and open-ended way, and to talk to the

users (Hix, 1996). From such naturalistic observations the researcher might proceed to a classification or taxonomy of the acts and errors that can be reliably observed, and that occupy users' time. The next step would be to count the number of failures of each kind over a sample of performance, and/or measure the times required to perform actions or complete subtasks.

F. SIMKIT

As previously stated THORN can display discrete event simulations as layers. The following provides the fundamental components of a discrete event simulation and discusses Simkit, the simulation package used by THORN.

There are two fundamental components of a discrete event simulation model (Buss, 1996). These are a set of state variables and a set of events. The discrete event model replicates the modeled systems behavior by producing state trajectories, or time plots, of the values of the modeled system's state variables. Measures of performance are determined as statistics of these state trajectories. Discrete event simulations are characterized by state trajectories that are piecewise constant. That is, events only occur at discrete moments in time when the value of at least one state variable changes. These events are instantaneous; no simulated time passes when an event occurs. Simulated time passes only between the occurrences of events.

The timing of the occurrence of events is controlled by a mechanism called an Event List. This is simply a listing of future events that have been scheduled. When it is time for a scheduled event to occur an event notice is generated. The event notice is comprised of two pieces of information: the event being scheduled, and the time at which the event is to occur. This process of scheduling future events, and executing them continues until there are no more events to execute. As discussed earlier, THORN uses a simulation package called Simkit to manage the events displayed in the discrete event layer. Simkit is described next.

The Operations Research Curriculum at the Naval Postgraduate School (NPS) uses a Java-based discrete event simulation package, called Simkit. Simkit allows rapid development of discrete event simulation models (Buss & Stork, 1996). This tool provides the foundation for the movement simulation utilized in the evaluation of

THORN. The relationship of THORN, Simkit and OpenMap™ is as follows: Simkit produces purely non-visual simulation output; OpenMap™ displays GIS data as layers; and THORN converts Simkit output into OpenMap™ layer data and provides interactive simulation controls. A detailed description of OpenMap™ and the THORN discrete event layer is provided next.

III. MODEL FEATURES

This thesis developed a GIS/graphical movement simulation, which allows the analyst to visualize the effects of the movement of forces on a geographical display. It then tested the usability of the system in the context of a movement simulation. Without the display, the simulation outputs a long list of unit locations, which is not inherently user-friendly. Without visualization of the locations of the individual units involved in the simulation the results cannot easily be verified or used. When this location data is fused with data in a GIS, the results of this simulation can be easily seen, analyzed, and verified, by planners and analysts. A loose coupling of components allows the movement simulation to be written independently of the map display.

The software model and architecture for this thesis is organized in the following manner:

1. A purely non-visual movement simulation which uses Simkit,
2. A geographical map display tool, capable of showing geographically related data on a map display.

As mentioned previously, the graphical display part of THORN is built on OpenMap™, an open source display tool.

A. OPENMAP™

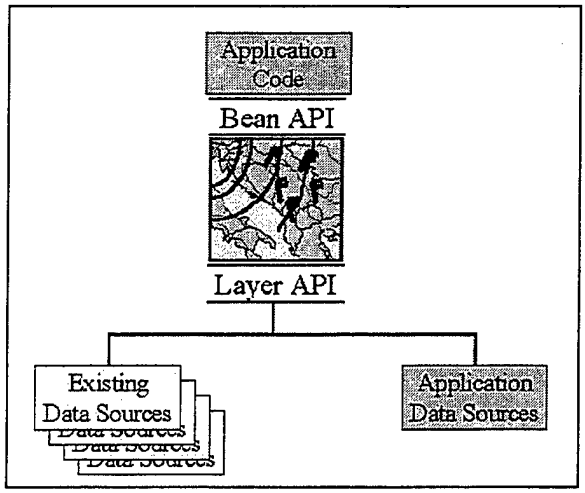


Figure 5 OpenMap™ Architecture Overview

OpenMap™ is a Java Beans™ based toolkit for building applications and applets needing geographic information. OpenMap™ components allow access to data from legacy applications in-place in a distributed setting. The Swing components in OpenMap™ understand geographic coordinates, help display map data, and help handle user input events to manipulate that data. The maps consist of graphical objects that can react to user inputs. The layers are Java Swing components that are totally responsible for drawing themselves as part of a greater whole map. Java Swing handles the layering of graphics from multiple layers. The independence of the layers gives them great leeway on how they can access their data source and create their graphics for the map. Layers can act as clients, creating graphics from data received from a server, or simply displaying graphics acquired from a server. They can also create graphics from internal algorithms.

The OpenMap™ architecture has the mechanism to dispatch mouse and keyboard events to Layers that want to receive them. Each Layer has the capability to change a graphic's appearance, add or delete graphics, or provide more information about a graphic. The graphic's information can be displayed, via the Information Delegator, in a Web browser, a text line section, or a pop-up window. At the heart of the architecture is the MapBean. The MapBean is a Swing component that is a map window. To define

what the map should look like, the MapBean needs a Projection. The Projection has a scale, a center latitude and longitude, a window pixel height and width, and a projection type. All of these attributes work towards describing the map in the MapBean window.

To place graphics on the map, layers need to be added to the MapBean. Layers create graphics from a data source and are notified when any attribute of the MapBean's projection is changed. They are expected to modify their list of graphics according to the parameters of the projection, pass the projection to the graphics so they draw themselves, and finally pass the Java Graphics (received in the layer's paint method) object to the graphics so they can draw themselves into it. The layers control how and (approximately) when their contributions to the map are drawn.

1. Components

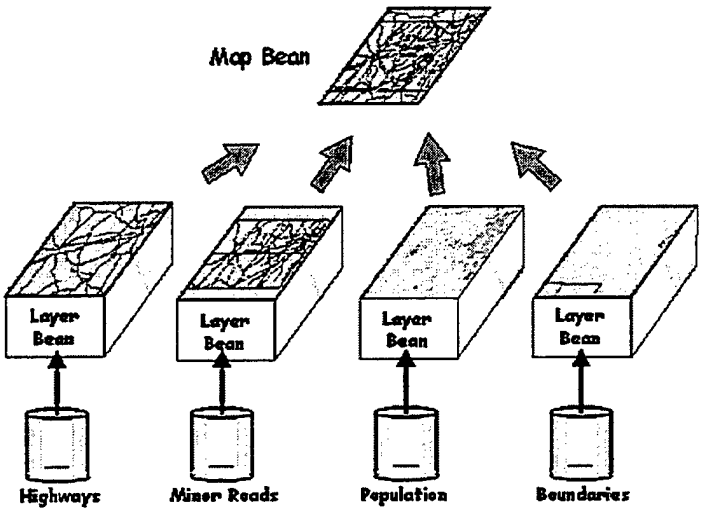


***Figure 6* MapBean Architecture**

a) MapBean

The MapBean, shown in Figure 6, is a Java Swing component that represents a map window. It holds a reference to the projection object, which is the description of how the map should be drawn (latitude/longitude location, scale, projection type, pixel height and width of the map). The MapBean is also the parent class to the layer objects, which act as child components to the MapBean.

The map can be changed by modifying the projection that the MapBean has or by changing the layers that are contained within the MapBean. The layers listen for any changes to the projection in the MapBean, update their graphics accordingly, and then redraw themselves.

b) Projections

OpenMap™ has Mercator, Orthographic, and Gnomic projections, as defined by the USGS Projection Manual 1932. There is also a CADRG projection, an Albers Equal Arc projection that is compliant with the pixel spacing defined in NIMA's Raster Product Format (RPF) specification. Finally there is a simple XY projection.

OpenMap™ projections are able to do more than forward and inverse translations - they are capable of defining these functions for different shape types, attempting to resolve some of the ambiguities of drawing these graphics on a globe. The Projection interface allows users read-only access to the current MapBean projection. The MapBean updates all the Layers and other ProjectionListeners when the view changes. A Projection object is defined with:

- Latitude and longitude of the center point of the MapBean canvas
- Scale
- Height and width of the MapBean canvas
- Projection type (Mercator, Orthographic, etc)

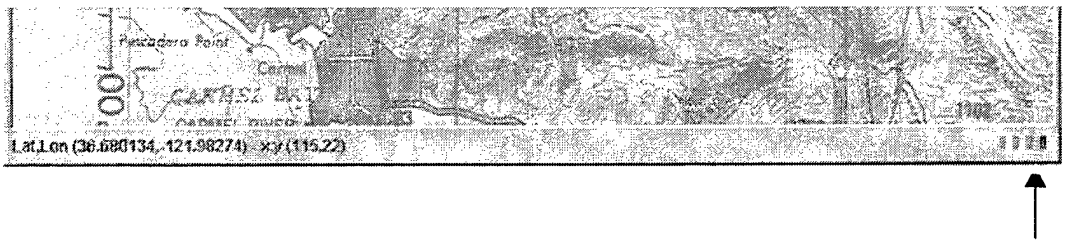


Figure 7 Status lights

c) *Information Delegator*

The Information Delgator is the object that directs messages to the user. It controls input to the text line at the bottom of the map, and has the ability to bring up a message window or a web browser to provide more information to the user. It listens to the layers that are active within the MapBean, and can display status lights, shown in Figure 7, (images) for each layer which indicate whether the layer is working on it's contribution to the map. The status lights are only functional if the layer is sending status updates to the Information Delegator.

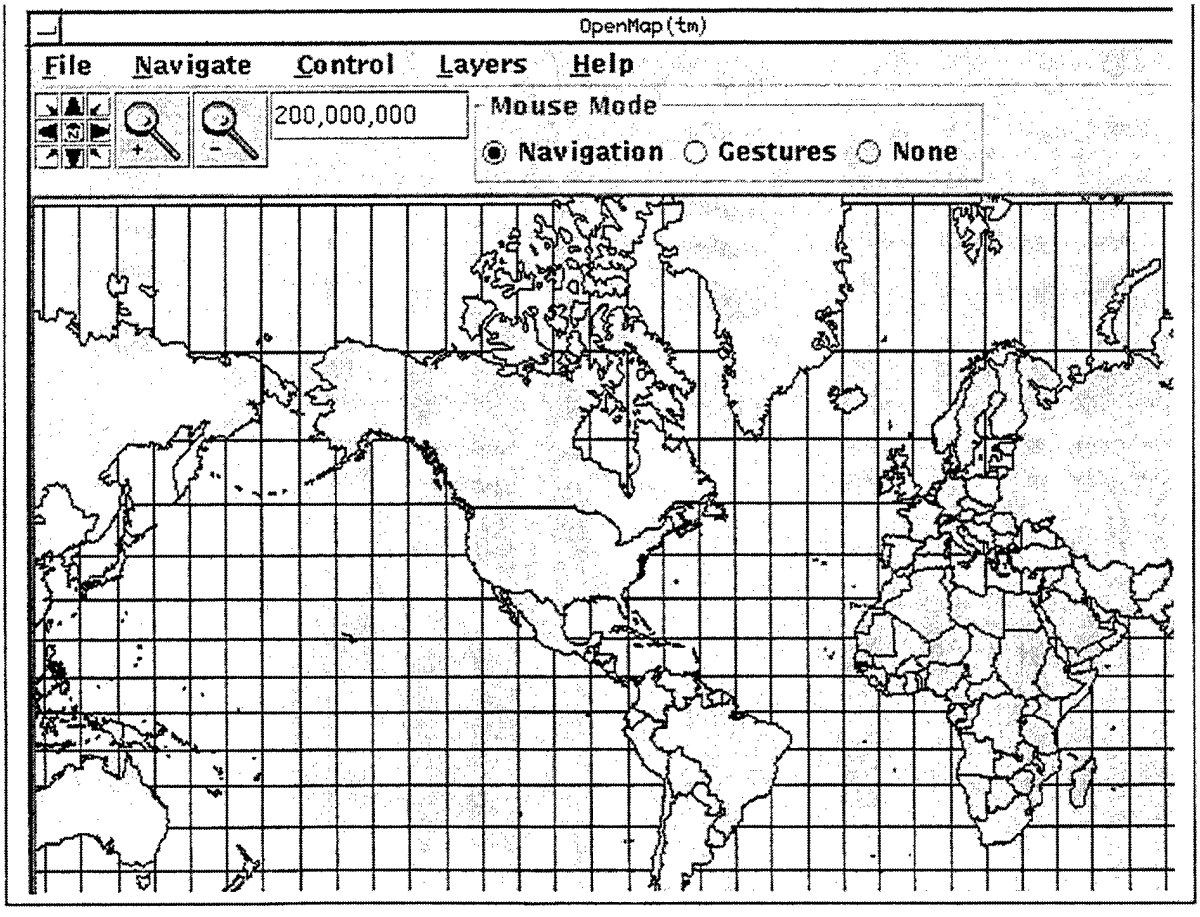


Figure 8 OpenMap™ Viewer Application

2. GUI

The OpenMap™ Viewer application starts with a map of the Earth with a 10° graticule, Figure 8. The following user interface items are available, Figure 9:

Figure 9 OpenMap™ Menu bar

a) Menu Items

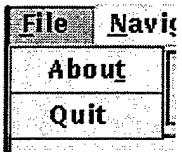
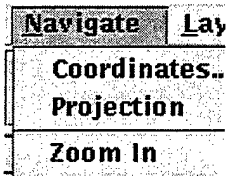


Figure 10 File menu item

(1) File. There are two menu items contained here, Figure 10. The *About* item provides information about the underlying OpenMap software, and th



es the map application.

Figure 11 Navigate menu item

(2) Navigate Menu. Menu items found under the Navigate menu, Figure 11, are: the Coordinates item which present a *Reposition map*

dialog; Projection which allows the user to set the map display projection; and the *Zoom In/ Zoom Out* controls which zoom the map by the specified amount.

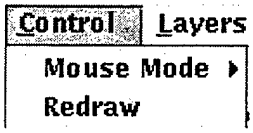


Figure 12 Control menu item

(3) Control Menu. The Control Menu, Figure 12, has two items. *Mouse Mode...* which changes the mouse behavior in the following manner. *Navigate* allows you to move around on the map. *Gestures* passes mouse events through to layers, and *None* ignores all mouse clicks. The *Redraw* redraws the map.

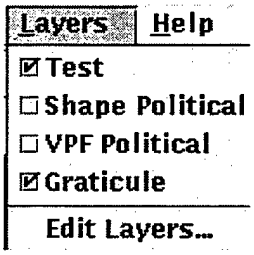


Figure 13 Layers menu item

(4) Layer Menu. The layer menu ,Figure 13, is where the user defined map layers are manipulated. It is arranged by the relative position of the layer in the display. The layer display status is controlled by selection of the check box. The *Edit Layers* item brings up the layer editor window.

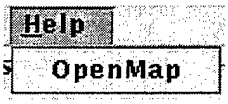


Figure 14 Help menu item

(5) Help Menu, Figure 14, contains a single menu item that will bring up help documents in a user specified web browser.

Figure 15 OpenMap™ tool bar

b) Toolbar Items

The following items can be found on the toolbar, Figure 15: *Rosette,* the rosette pans the map in the specified direction and the middle button recenters the view to the starting point; *Magnifying Glass,* "+" Zooms in 2X over the center of the map "-" Zooms out 2X over the center of the map; *Scale Entry* allows the user to enter the scale of the map; *Mouse Mode,* Changes the mouse behavior.

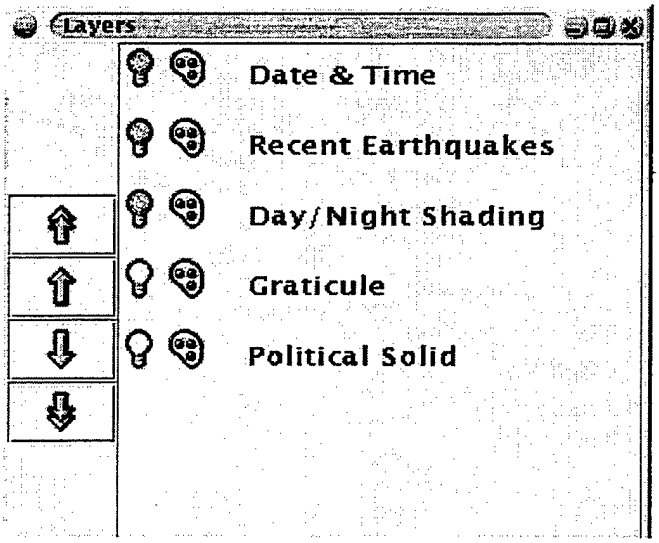


Figure 16 Layer editor palette

c) Layer Editor window

The layer editor window, Figure 16, uses the following icons to manipulate layers in the display: Turn layer off/on; turn layer palette (GUI) controls off/on; move selected layer to top of map; move selected layer up one level in map; move selected layer down one level in map; move selected layer to bottom of map.

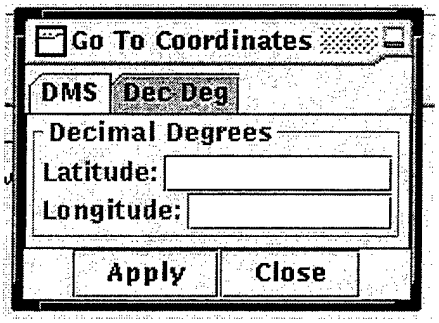


Figure 17 Coordinates editor window

d) Coordinates Window

The Coordinates Window, Figure 17, is used to specify coordinates in decimal degrees or degrees, minuets and seconds (DMS). Once the data is entered the user clicks apply and the map will be recentered over the position indicated.

3. Discrete Event Layer

THORN is the result of modifying the OpenMap application's event messaging algorithm to incorporate a animated discrete event layer. The discrete event layer (DEL) developed for THORN utilizes the a *Model-View-Controller* (MVC) paradigm to separate the simulation and display components. The MVC paradigm is an approach to programming that separates data input, data processing, and data output in such a way that either the input or the output can be modified without having any impact on the processing, see Figure 18.

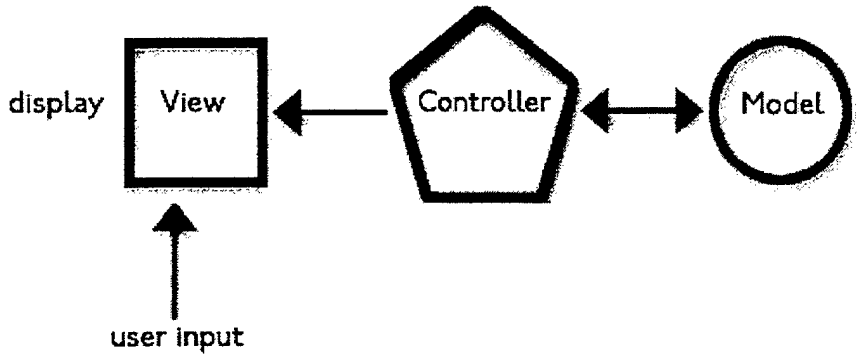


Figure 18 Model View Controller paradigm

THORN'S implementation of the MVC paradigm is as follows: the model is a discrete event simulation utilizing Simkit; the view is a geo-referenced icon representing the position of the system simulated in the model; the controller is a Simkit's mover manager that adjudicates model movement. In this arrangement the model has no interaction with the external view. This encapsulation allows a single view to display many models simultaneously. The benefit gained in this single view arrangement is the ability to have models developed orthogonally interact visually.

The DEL was implemented as a Layer Bean. It utilizes the OpenMap event-passing paradigm to communicate with the non-visual Simkit. The simulation is configured to manage entities whose maneuvering characteristics have been modeled utilizing Simkit's mover manager. The positions of the entities are then communicated to DEL via a Java runtime event. This position data is then geo-referenced and displayed in the DEL. THORN is capable of displaying and animating any Simkit entity. The visual representation of the entity is completely user configurable.

B. OMFTS SCENARIO

A very simple OMFTS scenario involving two battalion size forces was developed to evaluate the THORN interface. The scenario was designed to exercise THORN, and was not intended to illustrate all the features OMFTS. The Blue Force has made an amphibious landing and is using the existing road network to engage the opposition. The opposing force Red Force has been mobilized and will attempt to deploy and interdict the Blue Force. Two separate discrete event simulations model the force movement characteristics used in this scenario, each with its own set of movement rules. The Blue Forces are constrained to use the existing transportation network as a means of movement. The Red Forces have no constraints on movement networks, but must move in the general direction of the landing force.

	Task
1	"Turn on the DTED Layer"
2	"Turn on the Blue Force Layer"
3	"Turn on the Red Force Layer"
4	"Change mouse to gesture mode"
5	"What is the elevation of the Blue Force"
6	"What is the elevation of the Red Force"
7	"Turn off the DTED Layer"
8	"Turn off the Blue Force Layer"
9	"Turn off the Red Force Layer"
10	"Change mouse mode to navigation"
11	"Zoom the map in to 1:50,000"
12	"Locate the Naval Postgraduate School"
13	"Turn on the Draw Tools Layer"
14	"Select the Draw Tools palette tool"
15	"Draw red a rectangle around the Naval Postgraduate School. Use Lat1: 36.95784 & Long1: -121.87584; Lat 2: 36.59676 & Long2: -121.87309"
16	"Fill the rectangle with a white fill color"
17	"Close the rectangle tool window"
18	"Place a text label, with the caption Naval Postgraduate School at Lat: 36.95784 & Long: -121.87584"
19	"Close all Draw Tools palettes"
20	"Zoom out to 1:250,000."
21	"Turn on the Blue Force Layer"

Figure 19 Scenario task script

Fused with the forces' positional data is elevation data obtained from Digital Terrain and Elevation Data (DTED), vector maps, and raster maps displayed as appropriate. The DTED are not used in the computations for the movement of forces, but since the discrete event model is just a component in this overall simulation system, these data could be dynamically fed to the model for use.

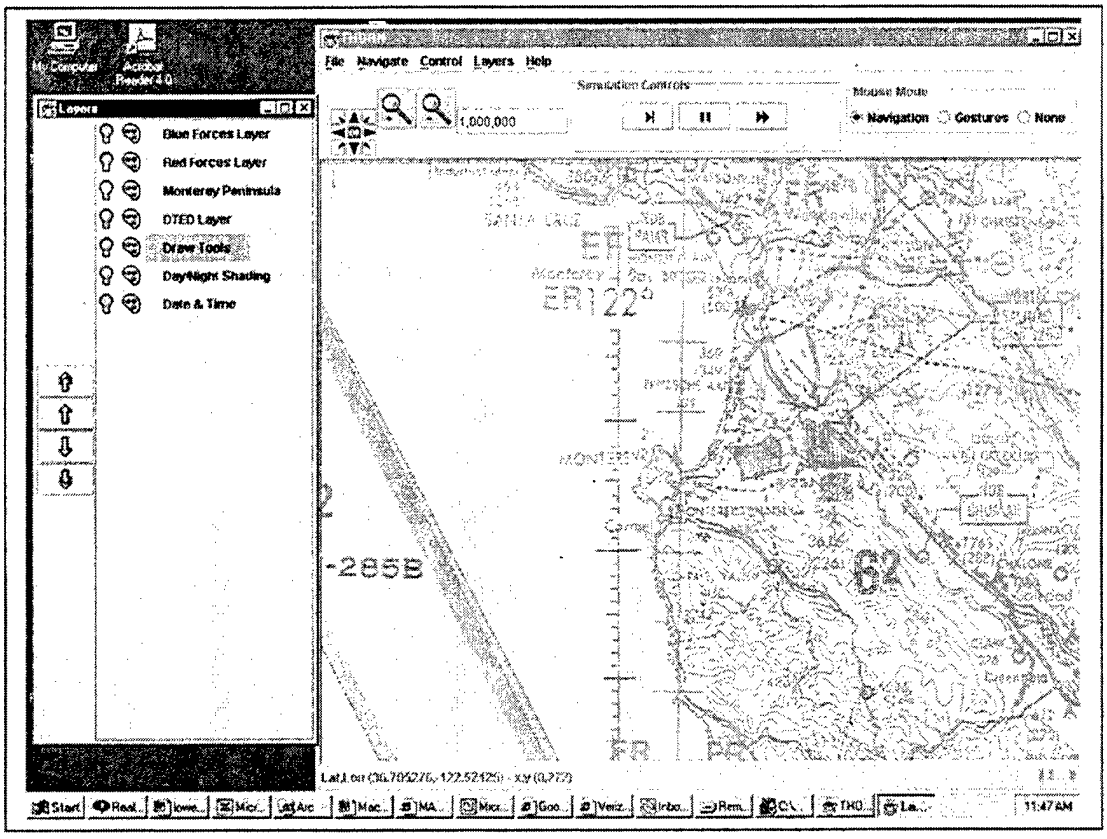


Figure 20 Scenario start

The task scenario begins with the application configured as seen in Figure 20. The user is shown a small-scale view of the Monterey peninsula in THORN'S main viewer application, and in a separate window all of the available layers are presented. Once the participant has oriented himself, he is instructed to turn on additional data layers and rescale the display.

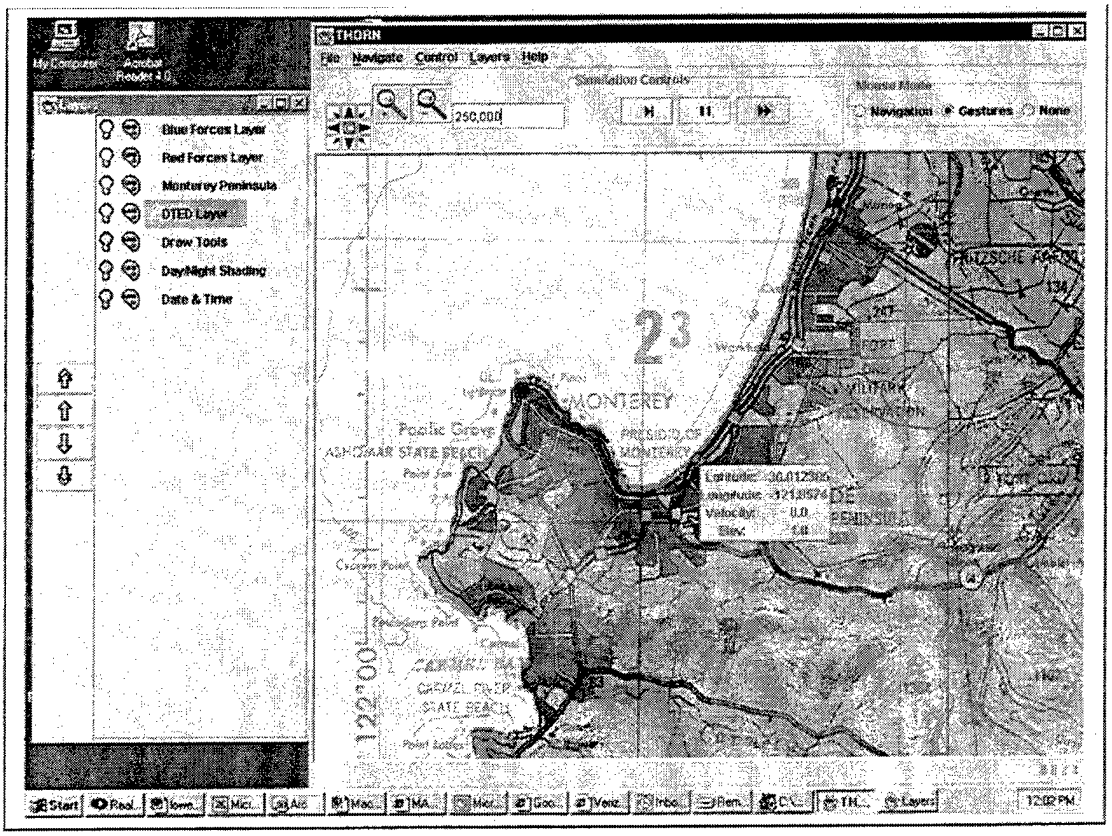


Figure 21 Scenario middle phase

As seen in Figure 21, the next set of exercise tasks involved interacting with the available information. In this case the participant queried the Blue force icon and is presented with tactical information gathered from the GIS. In this instance elevation data for the units are provided by a digital terrain elevation data (DTED) layer. The DTED information is also used in a shading algorithm that provides additional visual cues for elevation. If this information were not available the simulation would continue unhindered.

The next set of tasks involves annotating the display. These tasks require the user to locate and annotate landmark data on the display as shown in Figure 22. THORN treats the user's annotations as an additional semi-transparent layer. It updates the display and fuses the annotation with the original map data without destroying the underlying features of the map. This feature is extremely important and could not be done with traditional maps. Since the annotations are also geo-referenced the user could locate different map

data and use it with the existing annotations, or the user could simply print the current map, for distribution to the landing forces.

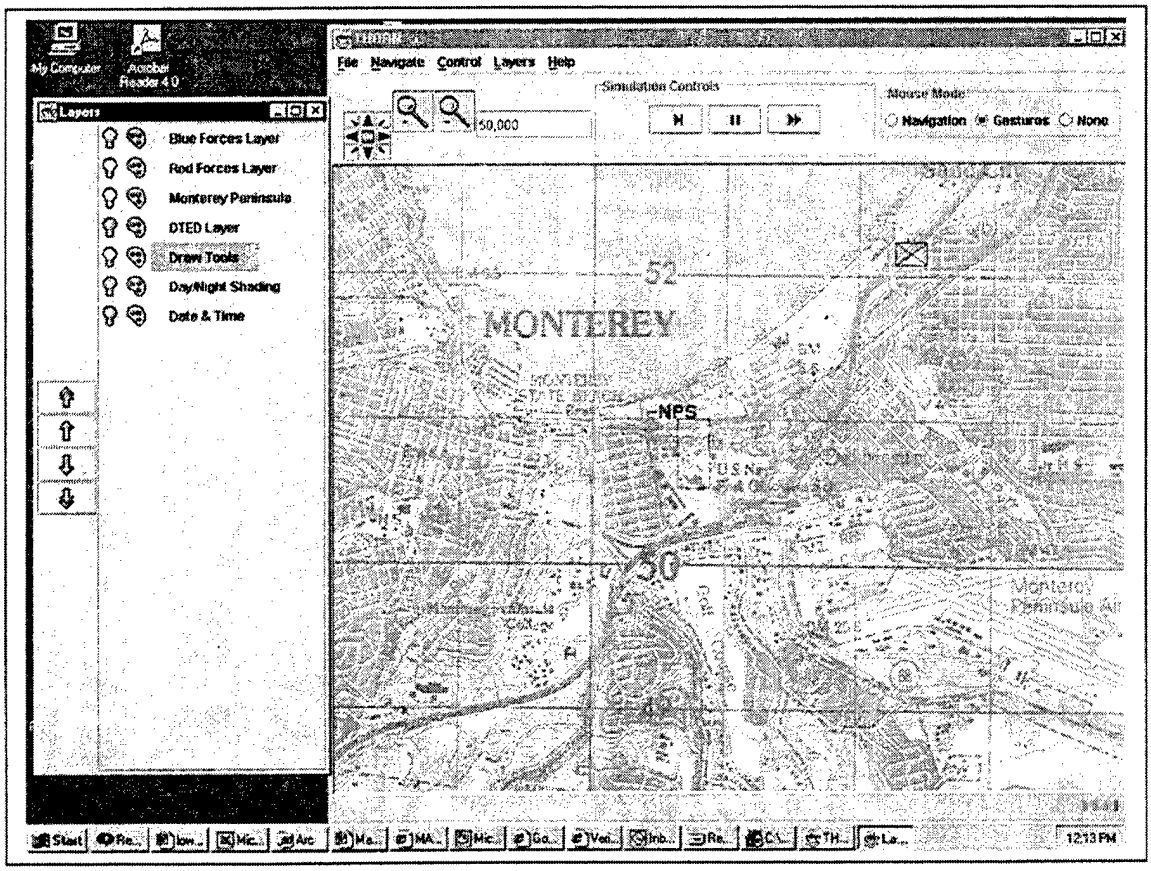


Figure 22 Annotation task

The final set of tasks, shown in Figure 23, involves the observation of a running discrete event simulation. Participants are tasked with starting the simulation and making observations of the activity that follows. This task evaluates the participant's ability to visually analyze the discrete event model. The model constrains the movement characteristics of the forces. The Blue Force must travel along the highway, while the Red forces are constrained to the Fort Ord operating area. The user can control the simulation by using simulation controls found on the toolbar, which start, pause, and restart the simulation. These commands are small subset of the capability of the model. It is important to note that the task scenario has the user complete a type of task before starting another. This single task execution scheme is not a limitation of the application. Annotation, navigation, and query can be performed at any time, even while the

simulation is running. The task scenario does not exercise this capability in an effort to avoid unnecessary confusion. The experiment methodology and results are discussed in the next chapter.

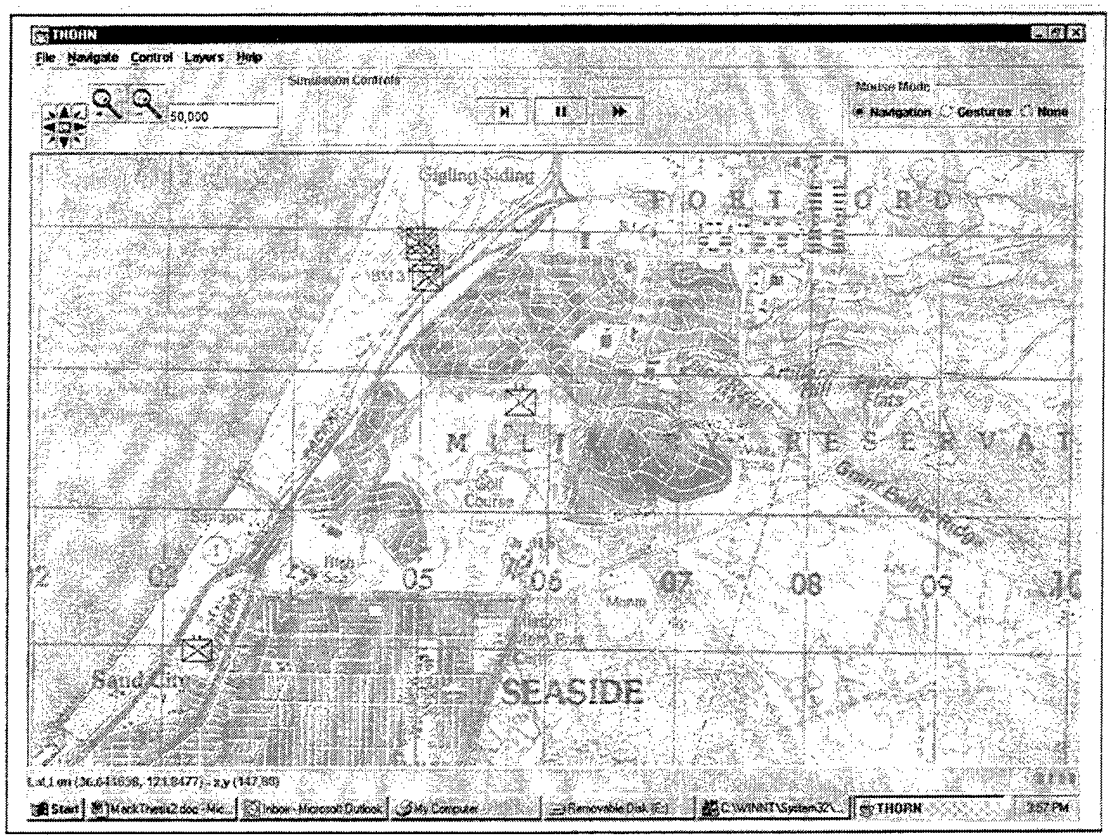


Figure 23 Scenario observation phase

IV. METHODOLOGY AND RESULTS

This chapter provides detail on the experimental design used in the evaluation of THORN'S usability. It also tests the hypothesis as to whether participant demographics are indicators of how well users can identify icons' functions.

A. RESEARCH APPROACH

This study involved the analysis of THORN. The purpose of this analysis was to assess the effect of the interface on the user and to identify any specific problems with the system.

Instrument. This study provides a benchmark across usability objectives. A usability task script and post-task questionnaire were administered to all subjects. The usability objectives of the overall study were:

- 90% Successful completion of tasks.
- 90% Error free rate.
- 90% score of 3 or better on a 7 point scale (e.g., 1=easy, 3=somewhat easy, 5=somewhat difficult, and 7=difficult) in ease-of-use.

Ideally, by the time an analytical system is released to the fleet, these objectives should be met and/or exceeded in order for the system to meet high ease-of use standards.

Procedure: participants completed an informed consent form and demographic questionnaire (Appendix A and B). The participants also received a usability task script along with a brief verbal description of the evaluation scenario (Appendix D). Participants sat directly in front of a 21-inch computer display monitor and controlled THORN with a computer mouse. The beginning of the usability evaluation consisted of the participants responding to a series of questions concerning their demographic background (Appendix C). Participants were then directed to read aloud and execute the tasks outlined in the task script. Following each task, questions concerning the usability of the THORN system were presented. Questions concerning participant satisfaction as well as current understanding of the THORN system were also presented.

Throughout each usability session, measurements were taken while the user performed tasks. These measurements were used to assess whether or not each usability objective had been met. These measurements were:

- *Task Completion Rate*: the proportion of participants who complete the task successfully and independently without critical errors. A critical error has occurred when the participant either requests assistance from the usability engineer or commits an uncorrected error that results in an incorrect outcome for the task.
- *Error Free Rate*: the proportion of participants completing the tasks without any errors, critical or non-critical. Non-critical errors include any error corrected by the test participant without intervention by the usability engineer or an error left uncorrected, but which does not affect the correctness of the outcome of the task.
- *User Satisfaction*: The User Satisfaction rating is derived from a series of questions which the user rates on a 7-point scale, ranging from very dissatisfied to very satisfied. The questions solicit user opinions with regard to ease-of-use, simplicity of the human-computer interaction, and system functionality.

B. DATA ANALYSIS

The occurrence of each of the measurements listed above was recorded in a spreadsheet. These data included any associated user-feedback information associated with the measurement. Frequencies of the various measurements in the database were determined, both in aggregate and by measurement type. The categorization of participants by experience level and whether they had previously used mapping tools was used in presenting the results. However, due to small sample size and no noticeable differences between categories, all subsequent analysis was performed on all participants as a single group.

The results of this usability evaluation are presented in the same order they were collected. The participant's demographic backgrounds are presented first, next are the participant's task completion rates, and finally the participant task satisfaction scores.

C. EFFECT OF PARTICIPANT DEMOGRAPHICS

Thirteen participants were used in the evaluation of THORN. The average participant's age was 32.5. All participants were male United States military officers in a postgraduate degree program. All participants possessed 20-20 correctable vision, and were physically able to operate the THORN application. It was hoped that the demographic survey would yield a model capable of indicating a participant's expectation of the function of a button by its icon. It was hypothesized that the participant's age, average time per session of computer use, and total number of hours per week spent on a computer would give insight in determining user expectations. However, this hypothesis could not be fully tested due to the homogeneity of the subject's demography.

As users spend more time on computers they develop expectations of toolbar icon function. For example, a button with a printer icon on a toolbar is assumed to be a shortcut for accessing the print command. As Schniderman states in rule two of the eight golden rules of interface design to maximize usability: enable frequent users to use shortcuts. The regression model is therefore:

$$\text{Score} = \beta_0 + \beta_1 \cdot \text{Length} + \beta_2 \cdot \text{Session} + \beta_3 \cdot \text{Years}$$

Figure 24 Abstract regression model

In this model Score is an indication of how well the icon indicates the application function, it is the frequency at which the user correctly identified button function by icon; Length is the average amount of time a user spends in front of a computer per session; Session is the number of times a person uses the computer each day; and Years the total number of years the participant has used a computer. These predictors were chosen since

they are a quantitative indication of the exposure the participant has had to computers. The assumption being, exposure to icons is proportional to the ability to discern the function represented.

Coefficient Estimates					
Label	Estimate	Std. Error	t-value	p-value	R Squared: 0.385713
Constant	1.13568	0.133733	8.492	0.0000	Sigma hat: 0.0438716
AGE	-0.00300208	0.00385963	-0.778	0.4517	Number of cases: 16
LENGTH	-0.000848667	0.000453119	-1.873	0.0856	Degrees of freedom: 12
SESS	-0.0127264	0.0110397	-1.153	0.2714	

Summary Analysis of Variance Table					
Source	df	SS	MS	F	p-value
Regression	3	0.0145024	0.00483414	2.51	0.1081
Residual	12	0.0230966	0.00192472		
Lack of fit	11	0.0190466	0.00173151	0.43	0.8456
Pure Error	1	0.00405	0.00405		

Figure 25 Regression Coefficients and Statistics

As shown in Figure 25, the regression was not statistically significant. This may be explained by the nature of the sample. The participants had very similar demographic data. The small sample size may have had an adverse effect as well.

Another explanation of the regression results may be that the demographic variables used in the regression model are simply not good predictors of an individual's ability to discern the functionality of THORN'S icons. This explanation can be substantiated by near homogenous computer literacy found in the participants of study. Computers have become so prevalent in the work place that the average participant could operate a computer efficiently, and could accomplish basic GIS tasks with ease even if he had never previously operated GIS software.

As stated in Sniderman's eight golden rules of interface design, usability can be maximized if the interface is targeted for a specific user group. The regression results seem to indicate that demographic data may not be a predictor of user expectation of button function based on icon, and cannot be used to match the interface to the user in this case. If this is the case, the results may mean that computer literacy and

pervasiveness have reached a level such that the majority of users can operate "industry standard" interfaces. The results may also indicate that interfaces that follow standard usability practices have a broader potential target audience. If this is indeed the case, then the results are more promising than a cursory analysis may reveal. In any case the results merit further research and data collection.

D. INITIAL IMPRESSIONS

Overall, participant's first impressions of the THORN interface were positive. Participants generally found THORN to be a familiar interface that contained more information than they were accustomed to in mapping systems, namely the inclusion of simulation controls. In addition, participants stated that their initial impression of the THORN interface was that it had a "standard" appearance; it used icon symbols that were commonly seen in software applications. Participants generally understood that the layer palette could be used to manipulate the contents of the viewer, and the scale indicator provided an indication of relative area coverage. Three participants did not know that they could manipulate the map scale by typing in values as well as using the zoom toolbar buttons. Generally participants figured out the navigation methodology, but two users never utilized the ability to center the map by clicking on it.

E. TASK COMPLETION

Once data was gathered on the participant's initial impressions of THORN and its components, a series of tasks (see Appendix C), in the form of a scenario, Figure 19,

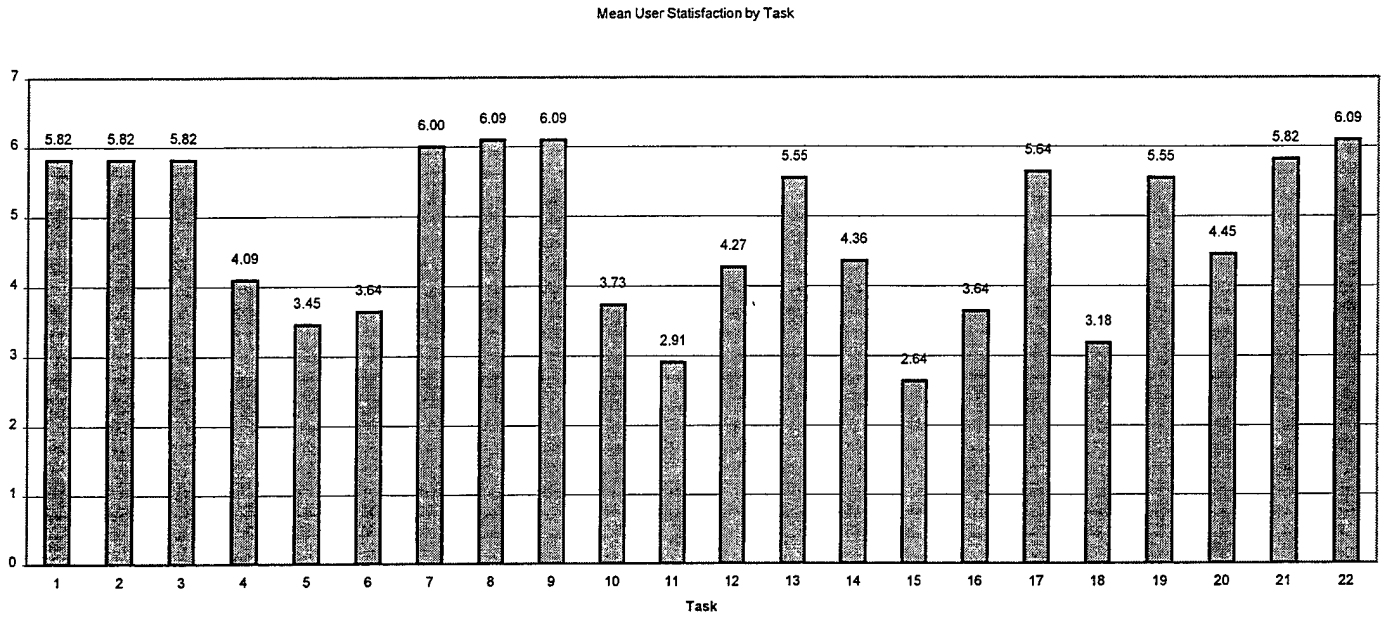


Figure 26 Mean User Satisfactions by Task

were presented. The task scenario contained four types of tasks: (1) layer display, which involved including and removing particular layers from the viewer; (2) layer interaction, tasks which required the user to query the layer for information or annotate a particular data point on the layer; (3) layer navigation, which involved navigating the layers to locate map information, and (4) general purpose tasks which are tasks required by all operating system GUIs. Task completion rates for all task types exceeded evaluation objectives. These rates were 91%, 100%, 97%, and 100% respectively. All evaluation participants completed these series of tasks without committing a critical error.

F. USER SATISFACTION

As seen in Figure 26, THORN meets the user satisfaction criteria of 90% score of 3 or better on a 7-point scale (e.g., 1=easy, 3=somewhat easy, 5=somewhat difficult, and 7=difficult) in ease-of-use. However if these data are compared with task completion rates, additional inferences can be made. The task group with the lowest task completion rate corresponds to the group of tasks with the highest user satisfaction scores. Tasks 1-3, 7-9, 21-22 are layer display tasks. The relatively low task completion rate can be attributed to initial user unfamiliarity with the THORN interface and layer concept. As shown in Figure 26, user satisfaction for this group of tasks is consistently higher than all other task groups. THORN provides redundant methods for accomplishing layer display tasks. It uses a menu based list of layers that can be turned on an off by simply selecting the desired layer, or the user can control layers by using the layer editor window, seen in Figure 16. By providing multiple means of accomplishing this task THORN can appeal to a larger number of users.

Task group (3) had the next highest user satisfaction rate. This group of tasks

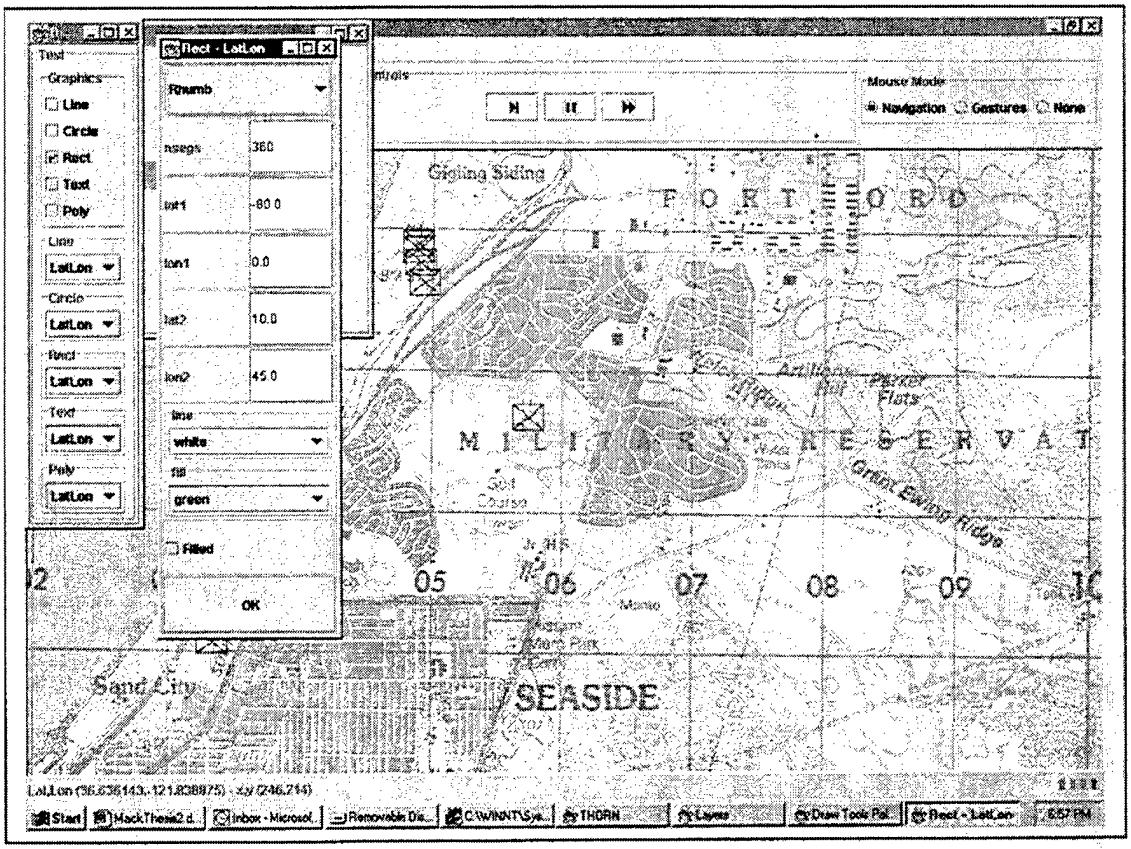


Figure 27 Annotation window

navigating the display and was exercised in tasks numbers 11 and 12. Once again THORN allows multiple ways for users to complete these tasks and achieved high satisfaction scores for task 12. Users could move to different locations by using the compass rosette found on the toolbar, or by clicking a location on the map, which causes THORN to re-center the map on the clicked location. Task 11 had a relatively low user satisfaction score, and this can be attributed to the implementation of the scale function. The current interface implementation does not give users the impression that they can type in a desired scale. This matter is further exasperated by the functionality of the zoom buttons. These buttons simply multiply or divide the value found in the scale window by a factor of two. So in some cases, the use of the buttons alone to scale the display to a desired level is not possible, i.e. desired scale is not a multiple of two. This implementation caused initial frustration.

Task groups (2) and (4) had perfect completion rates. Only task group (3) will be addressed here, since task group (4) is comprised of tasks implemented by the operating system and not THORN. Task group (3) is made up of tasks 5, and 6. It asked the user to interact with the display by querying units for information. THORN met the usability objective for these tasks but there is room for improvement. In order to interact with an icon the user must change to gesture mode, and click in the upper left corner of the icon. Participants routinely clicked in the center of the icon. While minor, correcting THORN to address the participant expectation could greatly increase satisfaction. Task 15 provided the largest potential for improvement. Its user satisfaction indicates a need for redesign. Task 15 asks the user to annotate a region on the map. The difficulty stems from the manner in which the annotation is performed. The user is required to manually enter coordinates into fields that detail the dimensions of the annotation, see Figure 27. This is an area for immediate attention in future versions of THORN. The current implementation is not consistent with the other elements of the interface, and is a potential source of data entry error. This type of error is very hard to locate and resolve. The following chapter summarizes the work, and provides recommendations for future work.

V. SUMMARY AND RECOMMENDATIONS

A. SUMMARY

THORN was developed to address some the shortcomings of current DoD simulation applications and to provide an OMFTS mission-planning tool. It allows users to visually analyze, verify, and validate information obtained from a discrete event simulation, and it was designed with usability in mind. This thesis evaluated THORN'S graphical user interface (GUI), and provided a baseline for future work. Specifically this thesis evaluated whether THORN'S interface meets the following usability standards:

- 90% Successful completion of tasks.
- 90% Error free rate.
- 90% score of 3 or better on a 7 point scale (e.g., 1=easy, 3=somewhat easy, 5=somewhat difficult, and 7=difficult) in ease-of-use.

The goals defined at the onset of this thesis were to produce a quality easy-to-use graphic user interface (GUI) for map-based mission planning and to conduct a usability test to determine its design success. As desired, THORN'S GUI evaluated favorably. It met or exceeded all evaluation objectives. However, there is room for improvement. Specifically, the annotation interface should be more consistent and interactive.

THORN successfully combines many of the proven tools from GIS software into a streamlined design while incorporating the strong design points of Human Factors guidelines. With continued GUI improvement and testing, THORN can grow to become a powerful and portable map-based mission-planning tool for OMFTS.

B. RECOMMENDATIONS

While THORN'S interface proved to be useful, there are areas that can be improved upon. Map annotation should be implemented in the same way drawing applications perform the draw task. THORN'S interface development should continue in a modular fashion. This modular development will allow the incorporation of new interface technologies without requiring complete interface redesign. As THORN'S

interface matures, periodic evaluations should be performed and compared against this benchmark to ensure THORN remains a valuable, and usable analytical tool.

Every attempt should be made to assemble demographic data that will provide good predictors of user expectation. Although the analysis in the study showed demographics had no significant impact on icon identification, more extensive studies may identify good demographic indicators. The next demographic survey should focus on the types of applications participants use, and use this data as predictors. As computers become more prevalent in the military work place, users will develop expectations of button function. Interface design should address this expectation by providing consistency *across* applications. It is not acceptable to redefine industry standard symbology.

Usability has to be incorporated from the beginning of the development process. The applications have to be small, flexible, extensible, and efficient. They must be capable of doing one task extremely well and sharing the results with everyone. These applications must be developed in Internet-time. If the application produces the solution one second late in a tactical environment, it has the same effect as if the application produced no solution at all. The compressed timeline of conflict and the rapid spread of the threat to US forces leave a very small margin for error.

APPENDIX A: CONSENT FORM

CONSENT FORM
Usability Evaluation of the THORN

Principal Investigator: LT Patrick Mack
Computer Science Department
Naval Postgraduate School
Monterey, Ca 93943

I, _____, consent to my participation in the research project titled Usability Evaluation of THORN.

I understand that I am free to withdraw my participation in the research at any time and that if I do I will not be subjected to any penalty of discriminatory treatment.

I have been given to opportunity to as questions about the research and received satisfactory answers.

I understand that any information or personal details gathered in the course of this research about me are confidential and that neither my name nor any other identifying information will be used or published without my written permission.

I understand that if I have any complaints or concerns about this research I can contact:
Arnold Buss
Operations Research Department
831-656-3259

Signed by: _____

Date

_____

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B: THORN SUBJECT QUESTIONNAIRE

1. What is your age? (20-25) (25-30) (30-35) (>35)

2. Male or Female? M F

3. Occupation? _____

4. If military, what rank and branch? _____

5. Highest Grade Completed?

   12 Assoc. BA/BS MA/MS MD/Ph.D.

6. Which is your dominate hand? Left Right

7. Are you currently experiencing any problems that impair your ability to use a computer?

   a) Yes b) No

If yes, what are they?_____

8. How many times do you use a computer a week? 1-5 5-10 10-15 >15

9. What is your most common computing session length?

   <10min 10-30min 30-60 min 60-90min >90min

10. How many sessions of this type do you have a day?

    a) 1
    b) 2
    c) 3
    d) 4
    e) > 4

11. Which of the following applications do you most often use on a daily basis? (circle as many as necessary)

a) Send / Receive e-mail
b) Surf the Internet
c) Word Processing
d) Finances
e) Spreadsheets
f) Games
g) Presentations
h) Programming
i) Other _____

12. What operating system do you primarily use? (circle more than one if needed)

a) Windows 9X, NT, 2K
b) Mac
c) Linux
d) Unix

13. How many years have you been actively using a computer?

a) < 1
b) 1-3
c) 3-5
d) 5-9
e) > 10 yrs.

14. Have you used map-based software? (commercial, military, Internet, etc.)

a) Yes
b) No

15. Are you geographically familiar with the Monterey Peninsula?

a) Yes
b) No

16. What is your attitude toward computer use?

a) Positive
b) Indifferent
c) Negative

APPENDIX C: THORN DATA COLLECTION SHEET

PART 1: ICONOGRAPHY

"What do you think the functions of the following icons are?

Icon	Correct (✓)	Incorrect (✓)	Accepted Answers
			pointer, arrow, mouse control
			move up/N
			move NE
			move right/E
			move SE
			move down/S
			move SW
			move left/W
			move NW
			center
			zoom in
			zoom out
			layer not loaded
			layer not available
			layer loaded
			stop
			pause
			play
			layer on
			layer off
			palette on
			palette off
			move layer to bottom
			move layer down one

PART 2: MAP MANIPULATION & INTERACTION

Function Tested	Task	Completed (✓)	Incomplete (✓)	Assistance Required (Y/N)
Layer Menu	"Turn on the DTED Layer"			
Layer Menu	"Turn on the Blue Force Layer"			
Layer Menu	"Turn on the Red Force Layer"			
Mouse Mode	"Change mouse to gesture mode"			
Interact	"What is the elevation of the Blue Force"			
Interact	"What is the elevation of the Red Force"			
Layer Menu	"Turn off the DTED Layer"			
Layer Menu	"Turn off the Blue Force Layer"			
Layer Menu	"Turn off the Red Force Layer"			
Mouse Mode	"Change mouse mode to navigation"			
Zoom In	"Zoom the map in to 1:50,000"			
Interact	"Locate the Naval Postgraduate School"			
Layer Menu	"Turn on the Draw Tools Layer"			
Layers Palette	"Select the Draw Tools palette tool"			
Layers Palette	"Draw red a rectangle around the Naval Postgraduate School. Use Lat1: 36.95784 & Long1: -121.87584; Lat 2: 36.59676 & Long2: -121.87309"			
Layers Palette	"Fill the rectangle with a white fill color"			
Layers Palette	"Close the rectangle tool window"			
Layers Palette	"Place a text label, with the caption Naval Postgraduate School at Lat: 36.95784 & Long: -121.87584"			
Layers Palette	"Close all Draw Tools palettes"			
Zoom	"Zoom out to 1:250,000."			
Layer Menu	"Turn on the Blue Force Layer"			
Layer Menu	"Turn on the Red Force Layer"			
Interact	"Observe the behavior of the entities"			
Interact	"Describe this behavior in the space provided below. Note movement characteristics and number of entities."			

APPENDIX D: FOLLOW-UP ICON RECOGNITION TEST

THORN v1.0a Usability Test Data Collection Sheet

Part Two

In two or three words, what do you think the functions of the following icons are?

Icon	Answer
	
	
	
	
	
	
	
	
	
	
	
	
	
	
	
	
	
	
	
	
	
	
	
	
	
	
	
	

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E: JAVA IMPLEMENTATIO OF A DISCRETE EVENT LAYER IN THORN

1) DesLayer – Implements discrete event simulation in THORN
2) DesIcon – Class used to display user depiction of unit icon
3) Generic Action – Implements event passing structure for animating events
4) PingEvent – Message passing paradigm for communication between objects
5) Layer – Generic GIS layer class

This appendix contains source code listings for the discrete event layer portion of THORN. Only source code critical to the development of the discrete event layer is presented here. The remaining source code is available upon request.

```
//Title:        DesLayer
//Version:      1.0
//Copyright:    Copyright (c) Pat Mack
//Author:       Pat Mack
//Company:
//Description:  Discrete Event Simulation Layer for OpenMap


import java.awt.Color;
import java.awt.event.*;
import java.awt.Point;
import java.awt.*;

import java.lang.reflect.Constructor;
import java.util.*;
import javax.swing.Box;
import javax.swing.Icon;
import javax.swing.ImageIcon;
import javax.swing.event.*;
import javax.swing.*;
import simkit.*;
import simkit.smd.*;

import com.bbn.openmap.LatLonPoint;
import com.bbn.openmap.Layer;
import com.bbn.openmap.layer.dted.DTEDCacheManager;
import com.bbn.openmap.event.*;
import com.bbn.openmap.util.*;
import com.bbn.openmap.omGraphics.*;
import com.bbn.openmap.proj.*;
import com.bbn.openmap.event.ProjectionEvent;
import PingSupport;
import PingListener;


/**
 * Layer that displays discrete event simulation.
 * This Layer is a PingAble (ActionListener) object so that it can be
 * prompted by a SimEvent. This layer understands the
 * following properties:
 * <code><pre>
 * # display icon as a Java ImageIcon
 * des.icon=Mover.gif
 * # display highlight icon as a Java ImageIcon
 * des.sicon=sMover.gif
 * #Ref to type of Mover that will be displayed
 * des.mover=simkit.smd.BasicMover
 * #Ref to Mover Manager
 * des.moverManager=RandomLocationManger
 * #Numer of entities to create
 * des.numMovers=1
 * #Initial Location of Mover
 * des.latLocation=
 * des.lonLocation=
 * #Velocity of Mover
 * des.velocity=.0005
 * <p>
```

```
 * In addition to the previous properties, you can get this layer to
 * work with the OpenMap viewer by adding/editing the additional
 * properties in your <code>openmap.properties</code> file:
 * <code><pre>
 * # layers
 * openmap.layers=des ...
 * # class
 * des.class=DesLayer
 * # name
 * date.prettyName=Discrete Event Layer
 * </pre></code>
 * NOTE: the color properties do not support alpha value if running on
 * JDK 1.1...
 */

public class DesLayer extends Layer implements
SimEventListener,MapMouseListener
{   //used to determine
    //private static final double DELTA = 0.015;

    // property keys

    public final static transient String iconProperty = ".icon";
    public final static transient String selIconProperty = ".sIcon";
    public final static transient String moverProperty = ".mover";
    public final static transient String mLatProperty = ".latMover";
    public final static transient String mLonProperty = ".lonMover";
    public final static transient String moverManProperty =
".moverMan";
    public final static transient String numMoversProperty =
".numMovers";
    public final static transient String velMoverProperty =
".velMover";


    // default properties
    private Properties myProps = null;
    private String myPrefix = "";
    private String iconString = "Mover.gif";
    private ImageIcon icon = new
     ImageIcon(DesLayer.class.getResource(iconString).getFile());
    private String sIconString = "sMover.gif";
    private ImageIcon sIcon = new
     ImageIcon(DesLayer.class.getResource(sIconString).getFile());
    private String moverClassString = "simkit.smd.BasicMover";
    private String manClassString = "RandomLocationMoverManager";
    private double iLatDouble = 31.57251;
    private double iLat = iLatDouble;
    private double iLonDouble = -97.10992;
    private double iLon = iLonDouble;
    private double iVelDouble = .0005;
    private double velocity = iVelDouble;
    private int iNumMoverInt = 1;
    private int numMovers = iNumMoverInt;
    private Properties props;

    private DTEDCacheManager dtedDelegator = null;
```

```
    private Mover theMover;


    //remaining fixed properties
    private OMGraphicList moverList;
    private OMGraphicList lineList;
    private Vector entities;
    private Projection proj;
    private PingSupport pingDelegator;
    public DesLayer() {

        moverList = new OMGraphicList();
        lineList = new OMGraphicList();
        Schedule.setVerbose(false);
        PingThread2.PT.addSimEventListener(this);


    }


    public void updatePosition() {

        int numMovers = moverList.size();
        fireStatusUpdate(LayerStatusEvent.START_WORKING);

        lineList.clear();
        for (int i = 0;i < numMovers ; i++) {
            ((DesIcon)moverList.getOMGraphicAt(i)).updatePosition();
            DesIcon temp = (DesIcon)moverList.getOMGraphicAt(i);
            LatLonPoint mPos = temp.getMoverLocation();
            LatLonPoint vPos = temp.getVirtualLocation();

        }

        if(proj != null){
        ((OMGraphicList)moverList).project((Projection)proj, true);
        }
        repaint();
        fireStatusUpdate(LayerStatusEvent.FINISH_WORKING);
    }

/**
 *  Here's where I hear the Ping event and update my entities.
**/


  public void processSimEvent(SimEvent e) {
    if (e.getEventName().equals("Ping")) {
       this.updatePosition();
    }
  }
```

```
    //------------------------------------------------------------------
-----
    // Layer overrides
    //------------------------------------------------------------------
-----


    /**
     * Renders the graphics list.  It is important to make this
     * routine as fast as possible since it is called frequently
     * by Swing, and the User Interface blocks while painting is
     * done.
     */
    public void paint(java.awt.Graphics g) {

      if(lineList.size() > 0){
        lineList.render(g);
      }
      moverList.render(g);
      fireStatusUpdate(LayerStatusEvent.FINISH_WORKING);
    }

    public void setProperties(String prefix, Properties theProps){

        Constructor NewMover[] = null;
        Constructor NewMoverMan[] = null;
        Mover temp = null;
        SimEntityBase temp2 = null;
        props = theProps;

        String tempProps = props.getProperty("openmap.layers");
        String[] paths = new String[tempProps.length()];
        int count = 0;
        StringTokenizer token = new StringTokenizer(tempProps);
        while(token.hasMoreElements()){
            if(token.nextElement().equals("jdted")){
              StringTokenizer dtedPaths = new
StringTokenizer(props.getProperty("jdted.paths"));
               while(dtedPaths.hasMoreElements()){
                    paths[count]=dtedPaths.nextToken();
                    count++;
               }
               dtedDelegator= new DTEDCacheManager(paths);
            }
        }

        super.setProperties(prefix,props);

        iconString = props.getProperty(prefix+iconProperty,iconString);
        sIconString =
props.getProperty(prefix+selIconProperty,sIconString);
        iLatDouble = Double.parseDouble(

props.getProperty(prefix+mLatProperty,Double.toString(iLatDouble)));
        iLonDouble = Double.parseDouble(

props.getProperty(prefix+mLonProperty,Double.toString(iLonDouble)));
```

```
        iVelDouble = Double.parseDouble(
         props.

getProperty(prefix+velMoverProperty,Double.toString(iVelDouble)));
        iNumMoverInt = Integer.parseInt(

props.getProperty(prefix+numMoversProperty,Integer.toString(iNumMoverIn
t)));


        moverClassString = props.
         getProperty(prefix+moverProperty,moverClassString);
        manClassString = props.
         getProperty(prefix+moverManProperty,manClassString);


        Object[] args = {new Coordinate(iLatDouble,iLonDouble)
              ,new Double(iVelDouble)};
       try{

        for(int i = 0;i < iNumMoverInt; i++){

            NewMover =
Class.forName(moverClassString).getConstructors();

            NewMoverMan =
Class.forName(manClassString).getConstructors();

            temp = (Mover)makeOne(NewMover,args);

            Object[] args2 = {temp};
            temp2 =(SimEntityBase) makeOne(NewMoverMan,args2);

            icon = new

ImageIcon(DesLayer.class.getResource(iconString).getFile());
            sIcon = new

ImageIcon(DesLayer.class.getResource(sIconString).getFile());
            DesIcon tempIcon = new DesIcon(icon,(BasicMover)temp);
            moverList.add(tempIcon);

         }
        }catch (ClassNotFoundException e){
            System.out.println("Class not found exception" + e);

        }
         this.updatePosition();


    }

    private Object makeOne(Constructor[] cons, Object[] args){
           Object temp = null;
           for(int j=0;j < cons.length;j++){

if(this.isConstructor(cons[j].getParameterTypes(),args)){
```

```
                    try{
                    temp = (SimEntityBase) cons[j].newInstance(args);
                    continue;
                    }catch(InstantiationException h){
                        System.out.println("Instantialtion exception" +
h);
                    }catch(java.lang.reflect.InvocationTargetException
f){
                         System.out.println("Invocation target
exception" + f);
                         System.out.println("Construtor: " + cons[j]);
                         System.out.println("Arguments: " + args[0]);
                    }catch(IllegalAccessException g){
                        System.out.println("Illegal Acces Exception" +
g);
                    }
            }
                 else{
                    //System.out.println("Des Layer");
                    //System.out.println("No valid constructor");
                 }
              }
              return temp;

    }
    private boolean isConstructor(Class[] params, Object[] args){
        int counter =1;
        if(params.length == args.length){
            for(int i =0; i < params.length ; i++){
                //System.out.println(params[i] + " " +
args[i].getClass());
                if(params[i].equals(args[i])){
                  counter++;
                }
                else if(params[i].isPrimitive()){
                  if (params[i].equals(Float.TYPE)) {
                          counter++;
                   } // if
                   if (params[i].equals(Integer.TYPE)) {
                           counter++;
                   } // if
                   if (params[i].equals(Double.TYPE)) {
                           counter++;
                   } // if
                   if (params[i].equals(Long.TYPE)) {
                           counter++;
                   } // if
                   if (params[i].equals(Boolean.TYPE)) {
                          counter++;
                   } // if
                   if (params[i].equals(Byte.TYPE)) {
                           counter++;
                   } // if
                   if (params[i].equals(Short.TYPE)) {
                           counter++;
                   } // if
                   if (params[i].equals(Character.TYPE)) {
```

```
                        counter++;
                    }  // if
                }  // if
            }//for

        }//if
        else{
            return false;
        }
        //System.out.println("isConstructor returned:" + (counter ==
params.length));
        return(counter == params.length);
    }



    //-----------------------------------------------------------------
-----
    // ProjectionListener interface implementation
    //-----------------------------------------------------------------
-----


    /**
     * Handler for <code>ProjectionEvent</code>s.  This function is
     * invoked when the <code>MapBean</code> projection changes.  The
     * graphics are reprojected and then the Layer is repainted.
     * <p>
     * @param e the projection event
     */
    public void projectionChanged(ProjectionEvent e) {
        proj = e.getProjection();
        //System.out.println("Got projection change");
          ((OMGraphicList)moverList).project(e.getProjection(), true);
          repaint();
    }

    //-----------------------------------------------------------------
-----
    // Mouse Events
    //-----------------------------------------------------------------
-----


    /**
     * Returns self as the <code>MapMouseListener</code> in order
     * to receive <code>MapMouseEvent</code>s.  If the implementation
     * would prefer to delegate <code>MapMouseEvent</code>s, it could
     * return the delegate from this method instead.
     * @return MapMouseListener this
     */
    public MapMouseListener getMapMouseListener(){
      return this;
    }
```

```
    /**
     * Return a list of the modes that are interesting to the
     * MapMouseListener.  The source MouseEvents will only get sent to
     * the MapMouseListener if the mode is set to one that the
     * listener is interested in.
     * Layers interested in receiving events should register for
     * receiving events in "select" mode.
     * <code>
     * <pre>
     *      return new String[1] {
     *          SelectMouseMode.modeID
     *      };
     * </pre>
     * <code>
     * @see NavMouseMode#modeID
     * @see SelectMouseMode#modeID
     * @see NullMouseMode#modeID
     */
    public String[] getMouseModeServiceList() {
      return new String[] {
          SelectMouseMode.modeID
      };
    }

    /**
     * Invoked when a mouse button has been pressed on a component.
     * @param e MouseEvent
     * @return true if the listener was able to process the event.
     */
    public boolean mousePressed(MouseEvent e) {
      if (Debug.debugging("DiscreteLayer")) {
          System.out.println("DiscreteLayer.mousePressed()");
      }
      return true;
    }

    /**
     * Invoked when a mouse button has been released on a component.
     * @param e MouseEvent
     * @return true if the listener was able to process the event.
     */
    public boolean mouseReleased(MouseEvent e) {
      if (Debug.debugging("Dragged")) {
          System.out.println("DiscreteLayer.mouseReleased()");
      }
      /*
      if(drag){
          drag=false;

          int count =0;
          Point p = e.getPoint();
          LatLonPoint latlon = proj.inverse(p);
          Coordinate p1 =
          new Coordinate(latlon.getLatitude(),latlon.getLongitude());

          for (Enumeration f = entities.elements();
f.hasMoreElements();) {
```

```
            Mover nextMover = (Mover) f.nextElement();
            if(nextMover.isSelected()){
              ((Mover)entities.elementAt(count)).moveTo(p1);
            }
            else{
              count++;
            }
          }
        }
        */
      return true;
    }

    /**
     * Invoked when the mouse has been clicked on a component.
     * The listener will receive this event if it successfully
     * processed <code>mousePressed()</code>, or if no other listener
     * processes the event.  If the listener successfully processes
     * mouseClicked(), then it will receive the next mouseClicked()
     * notifications that have a click count greater than one.
     * @param e MouseListener MouseEvent to handle.
     * @return true if the listener was able to process the event.
     */
    public boolean mouseClicked(MouseEvent e) {
        int index =
moverList.findIndexOfClosest(e.getX(),e.getY(),0.0005f);
        if(index == -1){  //nothing selected
            return false;
        }
        DesIcon tempIcon = (DesIcon)moverList.getOMGraphicAt(index);
        boolean selected = tempIcon.isSelected();

        updatePosition();
        if(selected){


((OMRaster)moverList.getOMGraphicAt(index)).setImageIcon(icon);
            ((DesIcon)moverList.getOMGraphicAt(index)).changeSelect();
        }else if(!selected){


((OMRaster)moverList.getOMGraphicAt(index)).setImageIcon(sIcon);
            ((DesIcon)moverList.getOMGraphicAt(index)).changeSelect();
        }
        JPopupMenu info = new JPopupMenu();
        JLabel latHeader = new JLabel("Latitude:  ");
        JLabel lonHeader = new JLabel("Longitude:  ");
        JLabel velocityHeader = new JLabel("Velocity:  ");
        JLabel elevHeader = new JLabel("Elev: ");
        Float tempf = new
Float(tempIcon.getMoverLocation().getLatitude());
        String temp = tempf.toString();
        JLabel lat = new JLabel(temp);
        tempf= new Float(tempIcon.getMoverLocation().getLongitude());
        temp = tempf.toString();
        JLabel lon = new JLabel(temp);
        tempf= new Float(tempIcon.getMover().getSpeed());
```

```
        temp = tempf.toString();
        JLabel vel = new JLabel(temp);
        JLabel elev;
        tempf = new Float(dtedDelegator.
         getElevation(tempIcon.getMoverLocation().getLatitude(),
         tempIcon.getMoverLocation().getLongitude()));
         temp = tempf.toString();
         if(tempf.floatValue() == -500f){
            elev = new JLabel("NA");
         }
        else{
            elev = new JLabel(temp);
        }
        info.setLayout(new GridBagLayout());
        GridBagConstraints c = new GridBagConstraints();
        c.gridy = 0;
        info.add(latHeader,c);
        info.add(lat,c);
        c.gridy++;
        info.add(lonHeader,c);
        info.add(lon,c);
        c.gridy++;
        info.add(velocityHeader,c);
        info.add(vel,c);
        c.gridy++;
        info.add(elevHeader,c);
        info.add(elev,c);
        info.show(e.getComponent(),e.getX(),e.getY());

        return true;
    }

    /**
     * Invoked when the mouse enters a component.
     * @param e MouseListener MouseEvent to handle.
     */
    public void mouseEntered(MouseEvent e) {
      if (Debug.debugging("DiscreteLayer")) {
          System.out.println("DiscreteLayer.mouseEntered()");
      }
    }

    /**
     * Invoked when the mouse exits a component.
     * @param e MouseListener MouseEvent to handle.
     */
    public void mouseExited(MouseEvent e) {
      if (Debug.debugging("DiscreteLayer")) {
          System.out.println("DiscreteLayer.mouseExited()");
      }
    }

    // Mouse Motion Listener events
    //////////////////////////////

    /**
     * Invoked when a mouse button is pressed on a component and then
```

```
     * dragged.  The listener will receive these events if it
     * successfully processes mousePressed(), or if no other listener
     * processes the event.
     * @param e MouseMotionListener MouseEvent to handle.
     * @return true if the listener was able to process the event.
     */
    public boolean mouseDragged(MouseEvent e) {

        if (Debug.debugging("Dragged")) {
              System.out.println("DiscreteLayer.mouseDragged()");
            }
        int index = moverList.findIndexOfClosest(e.getX(),e.getY());
        DesIcon tempIcon = (DesIcon)moverList.getOMGraphicAt(index);
        tempIcon.setVirtualLocation(e.getX(),e.getY());
        return true;
    }

    /**
     * Invoked when the mouse button has been moved on a component
     * (with no buttons no down).
     * @param e MouseListener MouseEvent to handle.
     * @return true if the listener was able to process the event.
     */
    public boolean mouseMoved(MouseEvent e) {
      if (Debug.debugging("DiscreteLayer")) {
          System.out.println("DiscreteLayer.mouseMoved()");
      }
      return true;
    }

    /**
     * Handle a mouse cursor moving without the button being pressed.
     * This event is intended to tell the listener that there was a
     * mouse movement, but that the event was consumed by another
     * layer.  This will allow a mouse listener to clean up actions
     * that might have happened because of another motion event
     * response.
     */
    public void mouseMoved() {
      if (Debug.debugging("DiscreteLayer")) {
          System.out.println("DiscreteLayer.mouseMoved()[alt]");
      }
    }









    //-----------------------------------------------------------------
-----
    // GUI
    //-----------------------------------------------------------------
-----

}
```

```
//Title:        DesIcon
//Version:      1.0
//Copyright:    Copyright (c) Pat Mack
//Author:       Pat Mack
//Company:
//Description:  Discrete Event Simulation Layer for OpenMap


import java.awt.Color;
import java.awt.event.*;
import com.bbn.openmap.event.*;
import javax.swing.*;
import simkit.smd.*;
import com.bbn.openmap.omGraphics.OMRaster;
import com.bbn.openmap.LatLonPoint;
import com.bbn.openmap.util.*;


public class DesIcon extends OMRaster {
    private static final Color selectColor = new Color(0.0f,1.0f,0.0f);
    private BasicMover vehicle;
    private boolean selected;
    private LatLonPoint virtualLocation;
    private LatLonPoint moverLocation;

    public DesIcon() {
        vehicle = new BasicMover(new Coordinate(),0);
        selected = false;
        this.setSelectColor(selectColor);
        moverLocation = new LatLonPoint(0.0f,0.0f);
        virtualLocation = moverLocation;


    }
    public DesIcon(ImageIcon image){
        super(0.0f,0.0f,image);
        vehicle = new BasicMover(new Coordinate(),0);
        selected = false;
        moverLocation = new LatLonPoint(0.0f,0.0f);
        virtualLocation = moverLocation;
    }
    public DesIcon(ImageIcon image, BasicMover mover){
        super((float)mover.getCurrentLocation().getXCoord(),
         (float)mover.getCurrentLocation().getYCoord(), image);
        vehicle = mover;
        selected = false;
        moverLocation =  this.getMoverLocation();
        virtualLocation = moverLocation;
        virtualLocation = moverLocation;
    }

    public boolean isSelected(){return selected;}
    public void changeSelect(){selected = !selected;}
    public void setVirtualLocation(float x, float y){
        virtualLocation = new LatLonPoint(x,y);
    }
```

```
    public LatLonPoint getVirtualLocation(){return virtualLocation;}
    public BasicMover getMover(){return vehicle;}
    public LatLonPoint getMoverLocation(){
        return new
LatLonPoint((float)vehicle.getCurrentLocation().getXCoord(),
         (float)vehicle.getCurrentLocation().getYCoord());
    }

    public DesIcon updatePosition(){
        float xposition =
(float)vehicle.getCurrentLocation().getXCoord();
        float yposition =
(float)vehicle.getCurrentLocation().getYCoord();
        this.setLat(xposition);
        this.setLon(yposition);
        return this;
    }


}
```

```
//Title:        GenericAction
//Version:      1.0
//Copyright:    Copyright (c) Pat Mack
//Author:       Pat Mack & Arnold Buss
//Company:
//Description:  Discrete Event Message Center


import javax.swing.*;
import java.awt.event.*;
import java.lang.reflect.*;

public class GenericAction extends AbstractAction {
   private Object target;
   private Method method;

   public GenericAction (Object theTarget, String methodName) {
      super(makeMenuName(methodName));
      target = theTarget;
      try {
         method = target.getClass().getMethod(methodName, null );
      }
      catch(NoSuchMethodException e) {
         System.err.println(e); e.printStackTrace(System.err);
           throw new
IllegalArgumentException(theTarget.getClass().getName() +
           " does not contain method " + methodName + "()");
      }
   }

   public GenericAction (Object theTarget, String methodName, Icon
icon) {
      super(makeMenuName(methodName), icon);
      target = theTarget;
      try {
         method = target.getClass().getMethod(methodName, null );
      }
      catch(NoSuchMethodException e) {
         System.err.println(e); e.printStackTrace(System.err);
         throw new
IllegalArgumentException(theTarget.getClass().getName() +
           " does not contain method " + methodName + "()");
      }
   }

   public GenericAction (Object theTarget, String methodName, Icon
icon,
     String tipText) {
      super(makeMenuName(methodName), icon);
      target = theTarget;
      this.putValue(Action.SHORT_DESCRIPTION, tipText);
      try {
         method = target.getClass().getMethod(methodName, null );
      }
      catch(NoSuchMethodException e) {
         System.err.println(e); e.printStackTrace(System.err);
```

```
            throw new
IllegalArgumentException(theTarget.getClass().getName() +
                " does not contain method " + methodName + "()");
        }
    }

    public GenericAction (Object theTarget, Method theMethod) {
        this(theTarget, makeMenuName(theMethod.getName()));
    }

    public void actionPerformed(ActionEvent event) {
        try {
            method.invoke(target,null);
        }
        catch( IllegalAccessException e ) {
            System.err.println(e); e.printStackTrace(System.err);
        }
        catch( IllegalArgumentException e ) {
            System.err.println(e); e.printStackTrace(System.err);
        }
        catch( InvocationTargetException e) {
            System.err.println(e.getTargetException());
            e.getTargetException().printStackTrace(System.err);
        }
    }

    public static String makeMenuName(String name) {
        if (name.endsWith("_")) {
            name = name.substring(0, name.length() - 1);
            name += "...";
        }
        name = name.replace('_', ' ');
        char[] chars = name.toCharArray();
        chars[0] = Character.toUpperCase(chars[0]);
        return new String(chars);
    }
}
```

```
//Title:        PingEvent
//Version:      1.0
//Copyright:    Copyright (c) Pat Mack
//Author:       Pat Mack & Arnold Buss
//Company:
//Description:  Discrete Event Simulation Layer for OpenMap


/**
 * An event to request that the simulation start or stop.
 *
 */
public class PingEvent extends java.util.EventObject
    implements java.io.Serializable
{

    public transient static final int START = 1;


    public transient static final int STOP = 0;

   // public transient static final PingT
    /**
     * The type of ping.
     */
    protected int type;


    /**
     * Construct a PingEvent.
     * @param source the creator of the PingEvent.
     * @param type the type of the event, refering to how to use the
amount.
     *
     */
    public PingEvent (Object source, int type)
    {
      super(source);
      switch (type) {
          case START:
          case STOP:
            break;
          default:
            throw new IllegalArgumentException("Invalid type: " +
type);
       }
      this.type = type;

    }

    /**
     * Check if the type is START.
     * @return boolean
     */
    public boolean isSTART ()
    {
      return (type == START);
```

```
    }

    /**
     * Check if the type is STOP.
     * @return boolean
     */
    public boolean isSTOP ()
    {
      return (type == STOP);
    }


    /**
     * Stringify the object.
     * @return String
     */
    public String toString ()
    {
      return new String("#<PingmEvent " +
                        (isSTOP() ? "Start " : "") +
                        (isSTART() ? "Stop " : "") +
                        ">");
    }
}
```

```
//Title:        PingListener
//Version:      1.0
//Copyright:    Copyright (c) Pat Mack
//Author:       Pat Mack & Arnold Buss
//Company:
//Description:  Discrete Event Simulation Layer for OpenMap


/**
 * Listens for requests to ping the simulation.
 */
public interface PingListener extends java.util.EventListener
{
    public void ping (PingEvent evt);
}
```

```
//Title:        PingSupport
//Version:      1.0
//Copyright:    Copyright (c) Pat Mack
//Author:       Pat Mack & Arnold Buss
//Company:
//Description:  Discrete Event Simulation Layer for OpenMap



import java.io.Serializable;
import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
import java.io.IOException;


/**
 * This is a utility class that can be used by beans that need support
 * for handling ZoomListeners and firing ZoomEvents.  You can use an
 * instance of this class as a member field of your bean and delegate
 * work to it.
 */
public class PingSupport implements java.io.Serializable {

    /**
     * Construct a PingSupport.
     * @param sourceBean The bean to be given as the source for any
events.
     */
    public PingSupport(Object sourceBean) {
      source = sourceBean;
    }

    /**
     * Add a PingSupport to the listener list.
     * @param listener The ZoomListener to be added
     */
    public synchronized void addPingListener(PingListener listener) {
      if (listeners == null) {
          listeners = new java.util.Vector();
      }
      listeners.addElement(listener);
    }

    /**
     * Remove a ZoomListener from the listener list.
     * @param listener The ZoomListener to be removed
     */
    public synchronized void removePingListener(PingListener listener)
{
      if (listeners == null) {
          return;
      }
      listeners.removeElement(listener);
    }
```

```
    /**
     * Send a zoom event to all registered listeners.
     * @param zoomType Either ZoomEvent.RELATIVE or ZoomEvent.ABSOLUTE
     * @param amount The new scale if ABSOLUTE, the multiplier if
RELATIVE
     */
    public void firePing(int PingType) {

      if (! ((PingType == PingEvent.START) ||
             (PingType == PingEvent.STOP))) {
          throw new IllegalArgumentException("Bad value, " + PingType +
                                      " for PingType in " +
                                      "PingSupport.firePing()");
      }

      java.util.Vector targets;
      synchronized (this) {
          if (listeners == null) {
            return;
          }
          targets = (java.util.Vector) listeners.clone();
      }
        PingEvent evt = new PingEvent(source, PingType);

      for (int i = 0; i < targets.size(); i++) {
          PingListener target = (PingListener)targets.elementAt(i);
          target.ping(evt);
      }
    }


    private void writeObject(ObjectOutputStream s) throws IOException {
        s.defaultWriteObject();

      java.util.Vector v = null;
      synchronized (this) {
          if (listeners != null) {
              v = (java.util.Vector) listeners.clone();
            }
      }

      if (v != null) {
          for(int i = 0; i < v.size(); i++) {
              PingListener l = (PingListener)v.elementAt(i);
              if (l instanceof Serializable) {
                  s.writeObject(l);
              }
            }
        }
        s.writeObject(null);
    }


    private void readObject(ObjectInputStream s) throws
ClassNotFoundException, IOException {
        s.defaultReadObject();
```

```
        Object listenerOrNull;
        while(null != (listenerOrNull = s.readObject())) {
        addPingListener((PingListener)listenerOrNull);
        }
    }

    transient private java.util.Vector listeners;
    private Object source;
    private int pingSupportSerializedDataVersion = 1;
}
```

```
/**
 *  This is an example of simple "behavior".  This MoverManager is
responsible
 *  for directing a single Mover.  The behavior is that a random
location is
 *  chosen in a rectangle determined by the instance variables
lowerLeft and
 *  upperRight, then the Mover is directed to proceed to that location.
Upon
 *  arrival, another random point is chosen and the Mover directed to
that.
 *
 *  @author Pat Mack
**/
import java.util.Vector;
import java.util.Enumeration;
import simkit.*;
import simkit.data.*;
import simkit.smd.*;
import simkit.util.*;

public class BlueForceMoverManger extends SimEntityBase {

  private static int location;
  private static final int NUM_MOVES=7;
  private static final double[] lat =
{36.612328,36.61471,36.61746,36.619843,
  36.62204,36.624603,36.636417};
  private static final double[] lon = {-121.854706,-121.85192,-
121.84914,
  -121.84707,-121.84563,-121.84138,-121.83046};


  private Mover myMover;
  private boolean cycling;
  private Coordinate destination;

  public BlueForceMoverManger(Mover m) {
    myMover = m;
    myMover.addSimEventListener(this);
    cycling = false;
    location = 0;
    destination = new Coordinate(lat[NUM_MOVES-1],lon[NUM_MOVES-1]);

  }


  public void startCycle() {
      cycling = true;
      myMover.moveTo(getNextLocation());
  }

  public void stopCycle() {
      cycling = false;
      myMover.stop();
```

```
  }

  public void doRun() {
      startCycle();
  }

  public void doEndMove(Mover m) {
     if (cycling) {
       startCycle();
     }
  }

  protected Coordinate getNextLocation() {
    location++;
    if(location < NUM_MOVES){
      return new Coordinate(lat[location],lon[location]);
    }
    else
      this.stopCycle();
      return destination;
  }

  public static void main(String[] args) {
    Mover vm = new BasicMover(new Coordinate(36.612328,-121.854706),
.0005);
    BlueForceMoverManger rlmm =
       new BlueForceMoverManger(vm);
    Schedule.setSingleStep(true);
    Schedule.stopOnTime(1000.0);
    Schedule.startSimulation();
  }
}
```

```
/**
 *  This is an example of simple "behavior".  This MoverManager is
responsible
 *  for directing a single Mover.  The behavior is that a random
location is
 *  chosen in a rectangle determined by the instance variables
lowerLeft and
 *  upperRight, then the Mover is directed to proceed to that location.
Upon
 *  arrival, another random point is chosen and the Mover directed to
that.
 *
 *  @author Arnold Buss
**/
import java.util.Vector;
import java.util.Enumeration;
import simkit.*;
import simkit.data.*;
import simkit.smd.*;
import simkit.util.*;

public class RedForceMoverManger extends SimEntityBase {

  private static int location;
  private static final int NUM_MOVES=9;
  private static final double LATLOWBOUND = 36.611;
  private static final double LONLOWBOUND = 121.815;
  private static final double LATHIBOUND = 36.664;
  private static final double LONHIBOUND = 121.857;
  private static final boolean NORTH_HEMI = true;
  private static final boolean WEST_HEMI =  true;

  private Mover myMover;
  private boolean cycling;
  private Coordinate destination;
  private long seed = 26751;
  private static UniformVariate latCoord;
  private static UniformVariate lonCoord;

  public RedForceMoverManger(Mover m){
      this(m,LATLOWBOUND,LATHIBOUND,LONLOWBOUND,LONHIBOUND);


  }
  public RedForceMoverManger(Mover m,
   double latLow, double latHigh, double lonLow, double lonHigh) {
    myMover = m;
    myMover.addSimEventListener(this);
    cycling = false;
    location = 0;
    lonCoord = new UniformVariate(lonLow,lonHigh,seed);
    latCoord = new UniformVariate(latLow,latHigh,seed);


  }
```

```
  public void startCycle() {
      cycling = true;
      myMover.moveTo(getNextLocation());
  }

  public void stopCycle() {
      cycling = false;
      myMover.stop();
  }

  public void doRun() {
      startCycle();
  }

  public void doEndMove(Mover m) {
     if (cycling) {
       startCycle();
     }
  }

  protected Coordinate getNextLocation() {
    int s1 = 0;
    int s2 = 0;
    if(NORTH_HEMI)
      s1=1;
    else
      s1=-1;
    if(WEST_HEMI)
      s2=-1;
    else
      s2=1;

     return new
Coordinate(s1*latCoord.generate(),s2*lonCoord.generate());
  }

  public static void main(String[] args) {
    Mover vm = new BasicMover(new Coordinate(37, -97), .05);
    RedForceMoverManger rlmm =
       new RedForceMoverManger(vm,35,37,-95,-97);
    Schedule.setSingleStep(true);
    Schedule.stopOnTime(50.0);
    Schedule.startSimulation();
  }
}
```

```
/*
**********************************************************************
 *
 *  BBNT Solutions LLC, A part of GTE
 *  10 Moulton St.
 *  Cambridge, MA 02138
 *  (617) 873-2000
 *
 *  Copyright (C) 1998, 2000
 *  This software is subject to copyright protection under the laws of
 *  the United States and other countries.
 *
 *
**********************************************************************
 *
 * $Source: /net/blatz/u4/rcs/openmap/com/bbn/openmap/Layer.java,v $
 * $Revision: 1.42 $
 * $Date: 2000/05/25 22:13:17 $
 * $Author: dietrick $
 *
 *
**********************************************************************
 */

package com.bbn.openmap;

import java.awt.*;
import java.awt.event.*;
import java.util.Vector;
import javax.swing.*;

import com.bbn.openmap.ProjectionPainter;
import com.bbn.openmap.event.*;
import com.bbn.openmap.proj.Projection;
import com.bbn.openmap.util.Debug;

/**
 * Layer objects are components which can be added to the MapBean to
 * make a map.
 * <p>
 * Layers implement the ProjectionListener interface to listen for
 * ProjectionEvents.  When the projection changes, they may need to
 * refetch, regenerate their graphics, and then repaint themselves
 * into the new view.
 */
public abstract class Layer
    extends JComponent
    implements ProjectionListener, ProjectionPainter
{

    /**
     * Precaches the swing package.  Computed based on the package of
     * <code>JComponent</code>.
     */
    protected static final String SWING_PACKAGE =
getPackage(JComponent.class);
```

```
/**
 * The listeners to the Layer that respond to requests for
 * information displays, like messages, requests for URL displays,
 * etc.
 */
protected Vector IDListeners = null;

/**
 * List of LayerStatusListeners.
 */
protected Vector lsListeners = null;

/**
 * Arguments modified by the Layer, or set by the Bean, at
 * runtime.
 */
protected String dynamicArgs = null;

/**
 * Flag to indicate whether a AWTToolkit is available. Almost
 * always should be left alone, unless you are doing something
 * without a display available.  This flag, when false, redirects
 * the repaint() method to fire a LayerStatusEvent.FINISH_WORKING
 * instead.
 */
protected static boolean AWTAvailable = true;

/**
 * Token uniquely identifying this layer in the application
 * properties.
 */
protected String markerName = null;

/**
 * Set AWTAvailable flag.
 * Your layer should not need to call this.
 * @param value boolean
 */
public static void setAWTAvailable(boolean value){
  AWTAvailable = value;
}

/**
 * Check AWTAvailable flag.
 * @return boolean
 */
public static boolean isAWTAvailable(){
  return AWTAvailable;
}

/**
 * Returns the package of the given class as a string.
 *
 * @param c a class
 */
protected static String getPackage (Class c) {
  String className = c.getName();
```

```
    int lastDot = className.lastIndexOf('.');
    return className.substring(0, lastDot);
  }


  /**
   * Override to only allow swing package listeners.  If Listeners
   * get added to the Layers, the mouse events don't make it to the
   * map.  Ever.
   * <p>
   * Swing popup menus, like <code>JPopupMenu</code> grab the
   * JComponent by adding themselves as <code>MouseListener</code>s.
   * So this method allows instances of classes in the xxx.swing
   * package to be added as <code>MouseListener</code>s, and no one
   * else.
   *
   * @param l a mouse listener.
   */
  public final void addMouseListener (MouseListener l) {
    String pkg = getPackage(l.getClass());
    if (java.beans.Beans.isDesignTime()) {
        super.addMouseListener(l);
    } else if (pkg.equals(SWING_PACKAGE)) {
        // Do nothing.  The menus work fine at the moment (5/19,
        // JDK 1.1.6, Swing 1.0.2), but may break in the future.
    } else if (pkg.startsWith(SWING_PACKAGE)) {
        // Do nothing.  This enables the menus to work
        // in JDK 1.2rc1, where the MouseListener is in
        // package javax.swing.plaf.basic and the SWING_PACKAGE
        // is javax.swing.
    } else {
        throw new IllegalArgumentException(
                "This operation is disallowed because the package \""
              + pkg + "\" is not in the swing package (\"" +
              SWING_PACKAGE + "\").");
    }
  }

  /**
   * Interface Layer method to get the dynamic args.
   * @return String args
   */
  public String getArgs () {
      return dynamicArgs;
  }

  /**
   * Interface Layer method to set the dynamic args.
   * @param args String
   */
  public void setArgs (String args) {
    dynamicArgs = args;
  }

  /**
   * Interface Layer method to receive layer arguments.
   * @param argv String[]
```

```
     */
    public void setArgs (String argv[]) {
    }

    /**
     * Accessor for the marker associated with this layer.  This is
     * the marker that uniquely identifies this layer in the
     * application properties.
     */
    public String getMarker() {
      return markerName;
    }

    /**
     * Sets the properties for the <code>Layer</code>.  This allows
     * <code>Layer</code>s to get a richer set of parameters than the
     * <code>setArgs</code> method.
     * Layers which override this method should do something like:
     * <code><pre>
     * public void setProperties (String prefix, Properties props) {
     *     super.setProperties(prefix, props);
     *     // do local stuff
     * }
     * </pre></code>
     * @param prefix the token to prefix the property names
     * @param props the <code>Properties</code> object
     * @see #setArgs
     */
    public void setProperties(String prefix, java.util.Properties
props) {
      setName(props.getProperty(prefix + ".prettyName", "Anonymous"));
      markerName = prefix;
    }

    /**
     * Returns the MapMouseListener object that handles the mouse
     * events.  This method is IGNORED in this class: it returns null.
     * Derived Layers should return the appropriate object if they
     * desire to receive MouseEvents.  The easiest thing for a Layer
     * to do in order to receive MouseEvents is to implement the
     * MapMouseListener interface and return itself.  A code snippet:
     * <code><pre>
     *  public MapMouseListener getMapMouseListener() {
     *          return this;
     *  }
     *  public String[] getMouseModeServiceList() {
     *          return new String[] {
     *              SelectMouseMode.modeID
     *          };
     *  }
     * </pre></code>
     * @return null
     */
    public synchronized MapMouseListener getMapMouseListener()
    {
      return null;
    }
```

```
    /**
     * Set the MapMouseListener for the layer.
     * This method is IGNORED in this class.
     * @param mml the object that will handle the mouse events for the
     * layer.
     * @deprecated this is an unnecessary function.  The Layer is
     * responsible for handling MouseEvents as it chooses.
     */
    public synchronized void setMapMouseListener(MapMouseListener mml)
    {
    }

    /**
     * Gets the gui controls associated with the layer.
     * This default implementation returns null indicating
     * that the layer has no gui controls.
     *
     * @return java.awt.Component or null
     */
    public java.awt.Component getGUI() {
      return null;
    }


    ////////////////////////////////////////////////
    //  InfoDisplay Handling Setup and Firing

    /**
     * Adds a listener for <code>InfoDisplayEvent</code>s.
     *
     * @param aInfoDisplayListener the listener to add
     */
    public synchronized void addInfoDisplayListener (
      InfoDisplayListener aInfoDisplayListener){
      if (IDListeners == null) {
          IDListeners = new java.util.Vector();
      }
      IDListeners.addElement(aInfoDisplayListener);
    }

    /**
     * Removes an InfoDisplayListener from this Layer.
     *
     * @param aInfoDisplayListener the listener to remove
     */
    public synchronized void removeInfoDisplayListener (
      InfoDisplayListener aInfoDisplayListener){
      if (IDListeners == null) {
          return;
      }
      IDListeners.removeElement(aInfoDisplayListener);
    }

    /**
     * Sends a request to the InfoDisplayListener to show the
information in
```

```
     * the InfoDisplay event on an single line display facility.
     * @param evt the InfoDisplay event carrying the string.
     */
    public void fireRequestInfoLine(InfoDisplayEvent evt){
      InfoDisplayListener temp[] = getSynchronizedListeners();
      if (temp != null){
          for (int i = 0; i < temp.length; i++){
            temp[i].requestInfoLine(evt);
          }
      }
      else Debug.message("Layer", getName() +
                         "|Layer.fireRequestInfoLine(): no info request
listener!");
    }

    /**
     * Sends a request to the InfoDisplay listener to display the
information
     * on an single line display facility.
     * The InfoDisplayEvent is created inside this function.
     * @param infoLine the string to put in the InfoDisplayEvent.
     */
    public void fireRequestInfoLine(String infoLine){
      fireRequestInfoLine(new InfoDisplayEvent(this, infoLine));
    }

    /**
     * Sends a request to the InfoDisplay listener to display the
information
     * in the InfoDisplay event in a Browser.
     * @param evt the InfoDisplayEvent holding the contents to put in
the
     * Browser.
     */
    public void fireRequestBrowserContent(InfoDisplayEvent evt){
      InfoDisplayListener temp[] = getSynchronizedListeners();
      if (temp != null){
          for (int i = 0; i < temp.length; i++){
            temp[i].requestBrowserContent(evt);
          }
      }
      else Debug.message("Layer", getName() +
                         "|Layer.fireRequestBrowserContent(): no info
request listener!");
    }

    /**
     * Sends a request to the InfoDisplayListener to display the
information
     * in a Browser.
     * The InfoDisplayEvent is created here holding the browserContent
     * @param browserContent the contents to put in the Browser.
     */
    public void fireRequestBrowserContent(String browserContent){
      fireRequestBrowserContent(new InfoDisplayEvent(this,
browserContent));
    }
```

```
    /**
     * Sends a request to the InfoDisplayListener to display a URL
given in
     * the InfoDisplay event in a Browser.
     * @param evt the InfoDisplayEvent holding the url location to give
to
     * the Browser.
     */
    public void fireRequestURL(InfoDisplayEvent evt){
      InfoDisplayListener temp[] = getSynchronizedListeners();
      if (temp != null){
          for (int i = 0; i < temp.length; i++){
            temp[i].requestURL(evt);
          }
      }
      else Debug.message("Layer", getName() +
                    "|Layer.fireRequestURL(): no info request
listener!");
    }

    /**
     * Sends a request to the InfoDisplayListener to display a URL in a
     * browser.
     * The InfoDisplayEvent is created here, and the URL location is
put
     * inside it.
     * @param url the url location to give to the Browser.
     */
    public void fireRequestURL(String url){
      fireRequestURL(new InfoDisplayEvent(this, url));
    }

    /**
     * Sends a request to the InfoDisplayListener to show a specific
cursor
     * over its component area.
     * @param cursor the cursor to use.
     */
    public void fireRequestCursor(java.awt.Cursor cursor){
      InfoDisplayListener temp[] = getSynchronizedListeners();
      if (temp != null){
          for (int i = 0; i < temp.length; i++){
            temp[i].requestCursor(cursor);
          }
      }
      else Debug.message("Layer", getName() +
                    "|Layer.fireRequestCursor(): no info request
listener!");
    }

    /**
     * Sends a request to the InfoDisplayListener to put the
information in
     * the InfoDisplay event in a dialog window.
     * @param evt the InfoDisplayEvent holding the message to put into
     * the dialog window.
```

```
     */
    public void fireRequestMessage(InfoDisplayEvent evt){
      InfoDisplayListener[] temp = getSynchronizedListeners();
      if (temp != null){
          for (int i = 0; i < temp.length; i++){
            temp[i].requestMessage(evt);
          }
      }
      else Debug.message("Layer", getName() +
                    "|Layer.fireRequestMessage(): no info request
listener!");
    }

    /**
     * Sends a request to the InfoDisplayListener to display the
information
     * in a dialog window.
     * The InfoDisplayEvent is created here, and the URL location is
put
     * inside it.
     * @param message the message to put in the dialog window.
     */
    public void fireRequestMessage(String message){
      fireRequestMessage(new InfoDisplayEvent(this, message));
    }

    /**
     * Get the InfoDisplayListeners.
     * Provides an internal InfoDisplayListener that is synchronized at
the
     * time of the check for null, so that we won't attempt to use it
     * later where there might have been an opportunity for it to have
     * been deleted.  Huh?
     * @return a personal copy of the InfoDisplayListener
     */
    protected InfoDisplayListener[] getSynchronizedListeners(){
      // use this for freakin' thread safety
      InfoDisplayListener[] temp = null;
      synchronized (this) {
          if (IDListeners == null) return temp;
          int numListeners = IDListeners.size();
          temp =  new InfoDisplayListener[numListeners];
          for (int i = 0; i < numListeners; i++){
             temp[i] = (InfoDisplayListener)IDListeners.elementAt(i);
          }
      }
      return temp;
    }


    ////////////////////////////////////////////////
    //  LayerStatus Handling Setup and Firing

    /**
     * Returns an array of all the LayerStatusListeners.
     * @return LayerStatusListener[]
     */
```

```
    protected LayerStatusListener[] getSynchronizedStatusListeners(){
      // use this for freakin' thread safety
      LayerStatusListener[] temp = null;
      synchronized (this) {
          if (lsListeners == null) return temp;
          int numListeners = lsListeners.size();
          temp =  new LayerStatusListener[numListeners];
          for (int i = 0; i < numListeners; i++){
             temp[i] = (LayerStatusListener)lsListeners.elementAt(i);
          }
      }
      return temp;
    }

    /**
     * Adds a listener for <code>LayerStatusEvent</code>s.
     *
     * @param aLayerStatusListener LayerStatusListener
     */
    public synchronized void addLayerStatusListener (
      LayerStatusListener aLayerStatusListener)
    {
      if (lsListeners == null) {
          lsListeners = new java.util.Vector();
      }
      lsListeners.addElement(aLayerStatusListener);
    }

    /**
     * Removes a LayerStatusListene from this Layer.
     *
     * @param aLayerStatusListener the listener to remove
     */
    public synchronized void removeLayerStatusListener (
      LayerStatusListener aLayerStatusListener){

      if (lsListeners == null) {
          return;
      }
      lsListeners.removeElement(aLayerStatusListener);
    }

    /**
     * Sends a status update to the LayerStatusListener.
     * @param evt LayerStatusEvent
     */
    public void fireStatusUpdate(LayerStatusEvent evt){
      if (AWTAvailable){
          LayerStatusListener[] temp =
getSynchronizedStatusListeners();
          if (temp != null){
            for (int i = 0; i < temp.length; i++){
                temp[i].updateLayerStatus(evt);
            }
          }
          else Debug.message("Layer", getName() +
```

```
                        "|Layer.fireStatusUpdate(): no
LayerStatusListener!");
      }
    }

    /**
     * Sends a status update to the LayerStatusListener.
     * @param evt LayerStatusEvent
     */
    public void fireStatusUpdate(int status) {
      fireStatusUpdate(new LayerStatusEvent(this, status));
    }

    /**
     * Repaint the layer.
     * You should not need to override this.
     */
    public void repaint(){
      if (AWTAvailable) super.repaint();
      else {
          // This looks like a fireStatusUpdate, right?  But that is
          // disabled if !AWTAvailable.  The only way to fire the
          // status is finished is by calling a repaint.  Doing
          // anything else confuses the GIFMapBean.  The firing of
          // this status update may be redundant for layers that use
          // the status updates already, but we have to play smart
          // for all layers, especially for those who don't play
          // nice.
          LayerStatusEvent evt = new LayerStatusEvent(this,
LayerStatusEvent.FINISH_WORKING);
          LayerStatusListener[] temp =
getSynchronizedStatusListeners();
          if (temp != null){
            for (int i = 0; i < temp.length; i++){
                temp[i].updateLayerStatus(evt);
            }
          }
      }
    }

    /**
     * Repaint the layer.
     * If you are using BufferedMapBean for your application,
     * WE STRONGLY RECOMMEND THAT YOU DO NOT OVERRIDE THIS METHOD.
     * This method marks the layer buffer so that it will be refreshed.
     * If you override this method, and don't call super.repaint(),
     * the layers will not be repainted.
     */
    public void repaint(long tm, int x, int y, int width, int height) {
        Component p = getParent();
      if(p instanceof BufferedMapBean) {
          ((BufferedMapBean)p).setRequestPaint(true);
          if (Debug.debugging("basic")) {
            Debug.output(getName() +"|Layer: repaint(tm=" + tm +
                         ", x=" + x +
                         ", y=" + y +
                         ", width=" + width +
```

```
                        ", height=" + height + ")");
            }
        }
        super.repaint(tm, x, y, width, height);
    }

    /**
     *  This method is here to provide a default action for Layers as
     *  they act as a ProjectionPainter.  Normally, ProjectionPainters
     *  are expected to receive the projection, gather/create
     *  OMGraphics that apply to the projection, and render them into
     *  the Graphics provided.  This is supposed to be done in the
     *  same thread that calls this function, so the caller knows that
     *  when this method returns, everything that the
     *  ProjectionPainter needed to do is complete.<P> If the layer
     *  doesn't override this method, then the paint(Graphics) method
     *  will be called.
     *
     * @param proj Projection of the map.
     * @param g java.awt.Graphics to draw into.
     */
    public void renderDataForProjection(Projection proj, Graphics g){
      paint(g);
    }


    /**
     * This method is called when the layer is added to the MapBean
     * @param cont Container
     */
    public void added(Container cont)
    {
    }

    /**
     * This method is called after the layer is removed from the
     * MapBean and when the projection changes.  We recommend that
     * Layers override this method and nullify memory-intensive
     * variables.
     * @param cont Container
     */
    public void removed(Container cont)
    {
    }
}
```

LIST OF REFERENCES

Bradley, G.H., Buss, A.H., An Architecture for Dynamic Planning Systems Using Loosely Coupled Components, Proposal for Reimbursable Research, Naval Postgraduate School Monterey, CA, USA, 1997.

Buss, A.H., Modeling with Event Graphs, Proceedings of the 1996 Winter Simulation Conference, D. Morrice, J. Charnes (eds), Coronado, CA, USA, 1996.

Buss, A.H., A Tutorial on Discrete-Event Modeling with Simulation Graphs, Proceedings of the 1995 Winter Simulation Conference, K. Kang, W. Lilegdon, D. Goldsman (eds), Arlington, VA, USA, 1995.

Chan, P., Lee, R., Kramer, D.,The Java Class Libraries, Second Edition, Volume 1, Addison-Wesley , Berkeley, CA, USA, 1998.

Chan, P., Lee, R., The Java Class Libraries, Second Edition, Volume 2, Addison-Wesley, Berkeley, CA, USA, 1997.

Chairman of the Joint Chiefs of Staff (1996). Joint Vision 2010, Pentagon, Washington, D.C.: Author.

Department of Defense (1999). Design Criteria Standard: Human Engineering, (DoD Publication No. MIL-STD-1472F), Washington, D.C.: Author.

Dix, A. J., Finlay, J. E., Abowd, G. D. & Beale, R. (1998). Human-Computer Interaction, Prentice Hall Europe.

Flanagan, D., Java in a Nutshell, A Desktop Quick Reference, Second Edition, O'Reilly, Sebasopol, CA, USA, 1997.

Hix, D. & Hartson, R. H. (1993). Developing User Interfaces Ensuring Usability Through Product & Process, John Wiley & Sons, Inc., New York, New York.

Law A.M., Kelton W.D., Simulation Modeling & Analysis, second edition, McGraw-Hill Inc., New York, NY, USA, 1991.

Nielsen, J. (1999). Heuristic Evaluation, Usability Inspection Methods, John Wiley & Sons, New York, New York.

Nielsen, J. (1993). Usability Engineering, Academic Press, Cambridge, Massachusetts.

Nielsen, J. & Molich, R. (1990). Improving a Human-Computer Dialog, Communications of the ACM, 33, 338-348.

Shneiderman, B. (1997). Designing the User Interface – Strategies for Effective Human-Computer Interaction, Third Edition, Addison-Wesley Longman Inc., Menlo Park, California.

Stork, K., Sensors in Object Oriented Discrete Event Simulation, Master Thesis, Naval Postgraduate School Monterey, CA, USA, 1996.

Weber, J.L., Using Java 1.2, Special Edition, QUE, Indianapolis, IN, USA, 1998.

Zukowski, J., Java, AWT reference, O'Reilly, Sebasopol, CA, USA, 1997.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. Prof. A.H. Buss, Code OR/Bu 3
   Naval Postgraduate School
   Operations Research Department
   Monterey, California 93943

4. Prof. Dan Boger, Code CS/Bo 3
   Naval Postgraduate School
   Computer Science Department
   Monterey, California 93943

5. Prof. R.P. Darken, Code CS/Dr 3
   Naval Postgraduate School
   Computer Science Department
   Monterey, California 93943

6. Prof. G.H.Bradley, Code OR/Bz 1
   Naval Postgraduate School
   Operations Research Department
   Monterey, California 93943

7. LT Patrick Mack 3
   169544 Water Gap Road
   Williams, Oregon 97544